Collaborative Software Design

Unit Test & Test Driven Development

Manuel Giffels (Manuel.Giffels@kit.edu)
KSETA Course — October 2025

Recap: Agile Methods

Recap: Agile Methods

The Agile Development Methodology encourages the developer to be:

- Flexible — adapt to new requirements
- Fast — make changes to the implementation on a short timescale
- Courageous — change key parts of the software frequently
- Ready — release versions of the software very often

How to ensure that your code is actually doing what you want?

Recap: Agile Methods

The Agile Development Methodology encourages the developer to be:

- Flexible — adapt to new requirements
- Fast — make changes to the implementation on a short timescale
- Courageous — change key parts of the software frequently
- Ready — release versions of the software very often

How to ensure that your code is actually doing what you want?

Luckily: The Agile Toolbox provides also techniques to support the developers!

Recap: Agile Methods

The Agile Development Methodology encourages the developer to be:

- Flexible — adapt to new requirements
- Fast — make changes to the implementation on a short timescale
- Courageous — change key parts of the software frequently
- Ready — release versions of the software very often

How to ensure that your code is actually doing what you want?

Luckily: The Agile Toolbox provides also techniques to support the developers!

Important: Using these techniques is important for a successful Agile development process

Otherwise, Agile might results in

- un-controlled
- un-coordinated
- un-necessarily wasted time

How do you ensure your program „works“?

How do you ensure your program „works“?

First consideration: What does „works“ mean?

- Completes processing in the required amount of time
- Returns the correct result
- Behaves appropriately in case of faulty input
- Does not crash
- Stays inside of the allocated memory and disk space limits

How do you ensure your program „works“?

First consideration: What does „works“ mean?

- Completes processing in the required amount of time
- Returns the correct result
- Behaves appropriately in case of faulty input
- Does not crash
- Stays inside of the allocated memory and disk space limits

At which stage are these checks necessary?

- During development
- Before a release
- When changing compiler/interpreter, runtime or external libraries

How do you ensure your program „works“?

First consideration: What does „works“ mean?

- Completes processing in the required amount of time
- Returns the correct result
- Behaves appropriately in case of faulty input
- Does not crash
- Stays inside of the allocated memory and disk space limits

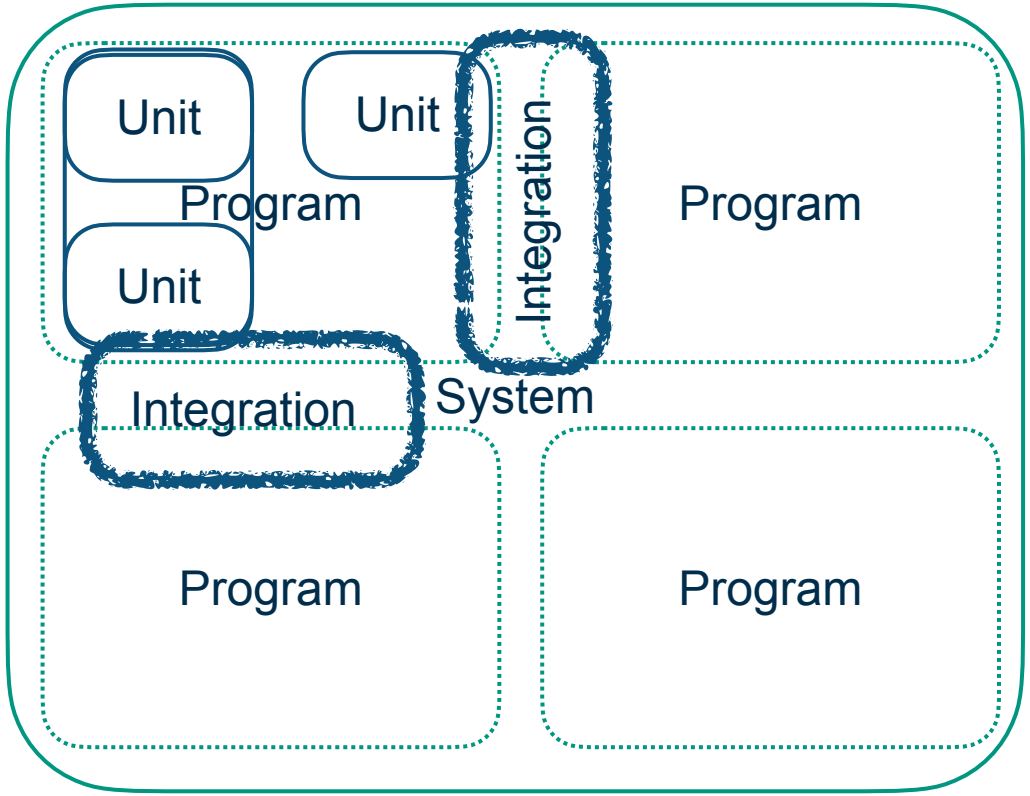
At which stage are these checks necessary?

- During development
- Before a release
- When changing compiler/interpreter, runtime or external libraries

Question: How do you ensure these points during your development process?

Test Scopes

- **Unittest**
  - Test part if the program
  - Check isolate behaviour, pinpoint defects in the software
- **Integration Test**
  - Test interaction with programs
  - Check outwards compatibility, ensure basic usability
- **System Test**
  - Test group of programs
  - Check requirements are met



Unit Testing

> „Unit tests are the tests that you write to verify the operation of your code. These tests aren't at the level of features. They are at the level of methods and functions."
>
> *Jeff Younker, Foundations of Agile Python Development, Apress, 2008*

Unit tests are:

- **Automated**
- **Compact** and test the smallest possible unit of functionality (a class, as method)
  in contrast to: Integration tests to validate the interplay between components
- **Fast**. The whole set of unit tests should complete in the order of seconds
- **Part of the code base** and available to every developer working on the project

Unit Test Frameworks

Unit test frameworks exist for every language and all offer a very similar set of features:

- A way to express test cases with a number of tests …
- … and group them together in so called test suites
- Method/function/macros to check for correct output of the tested source code
- Checks for error handling and exceptions
- Test runner: executing all or a sub-set of unit tests and report the result

Available frameworks:

- Python: unittest package (part of the python standard library), pytest
- C++: googletest, Boost.Test
- Java: JUnit

Anatomy of a Unittest

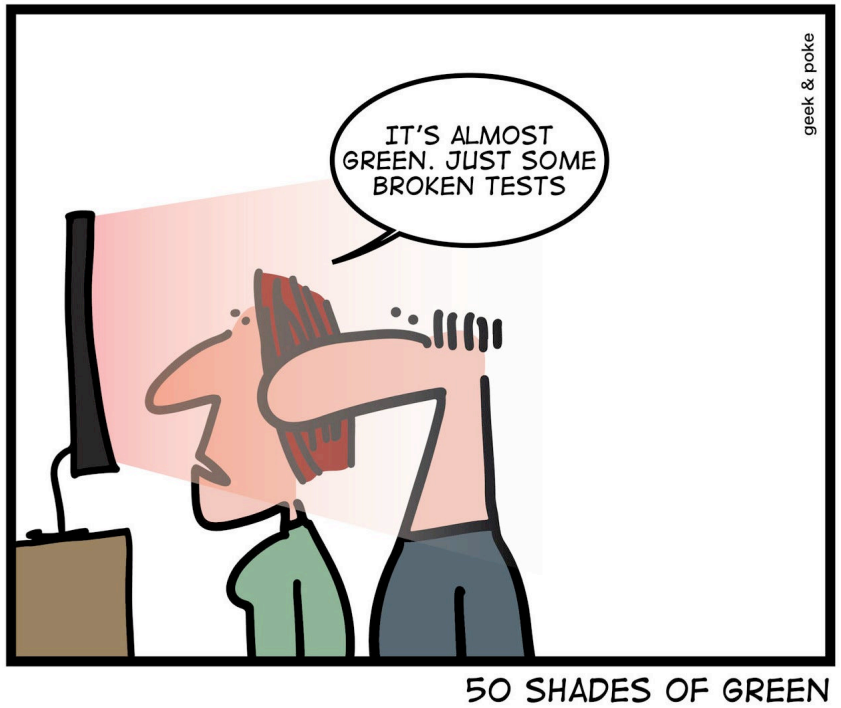
- What is a suitable „unit“?
  - Self-contained part of the code, e.g. a function
  - Expected/documented behaviour
  - Units may consist of other units
  - What would the users do? Unexpected input, etc.
- What is s suitable test?
  - Expected input/start → output/stop
    - Arbitrary selection of common cases
    - All edge cases you can think of
  - Relation to itself or other units
  - Sample of all guaranteed behaviour (includes reliable failure!)

```python
def divide(a, b):
    if not all(isinstance(x, (int, float)) for x in (a, b)):
        raise TypeError("Arguments must be numbers")
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero")
    return a / b
```

```python
class TestDivide(unittest.TestCase):
    def test_normal(self):
        self.assertEqual(divide(10, 2), 5)
    def test_zero_division(self):
        with self.assertRaises(ZeroDivisionError):
            divide(1, 0)
    def test_type_error(self):
        with self.assertRaises(TypeError):
            divide("10", 2)
```

Common Unit test Pitfalls

- Test behaviour, not implementation
  - Write tests that check what the code does, not how it does it internally
- Only testing the „happy path“
  - Missing edge cases, invalid input or exceptions
- Test depend on each other
  - Test should be independent and runnable in any order
- Be exhaustive but descriptive
  - Write a lot test cases to, but give them a reasonable naming
- Failure case must be identifiable
  - It should be immediately clear what went wrong and why
- Track expected failures
  - Sometimes a test is supposed to fail, document it and not just ignore it.



Python unit test example

The task: Implement a class to compute the average over multiple values:

```
class Average:
    def __init__(self):
        self._values = []

    def add_value(self, value):
        self._values.append(value)

    def compute(self):
        return sum(self._values)/len(self._values)
```

Question: What would you like to test in this implementation?

Python unit test example

A possible starting set of unit tests:

```
from unittest import TestCase
from average import Average

class TestAverage(TestCase):
    def setUp(self):  # execute once for every test
        self.average = Average()

    def test_zero_entries(self):
        with self.assertRaises(NoEntries):
            self.average.compute()

    def test_one_entry(self):
       self.average.add_value(23.0)
       self.assertEqual(23.0, self.average.compute())

    def test_three_entries(self):
        for value in (10.0, -20.0, 1.0):
            self.average.add_value(value)
        self.assertAlmostEqual(3, average,compute())
```

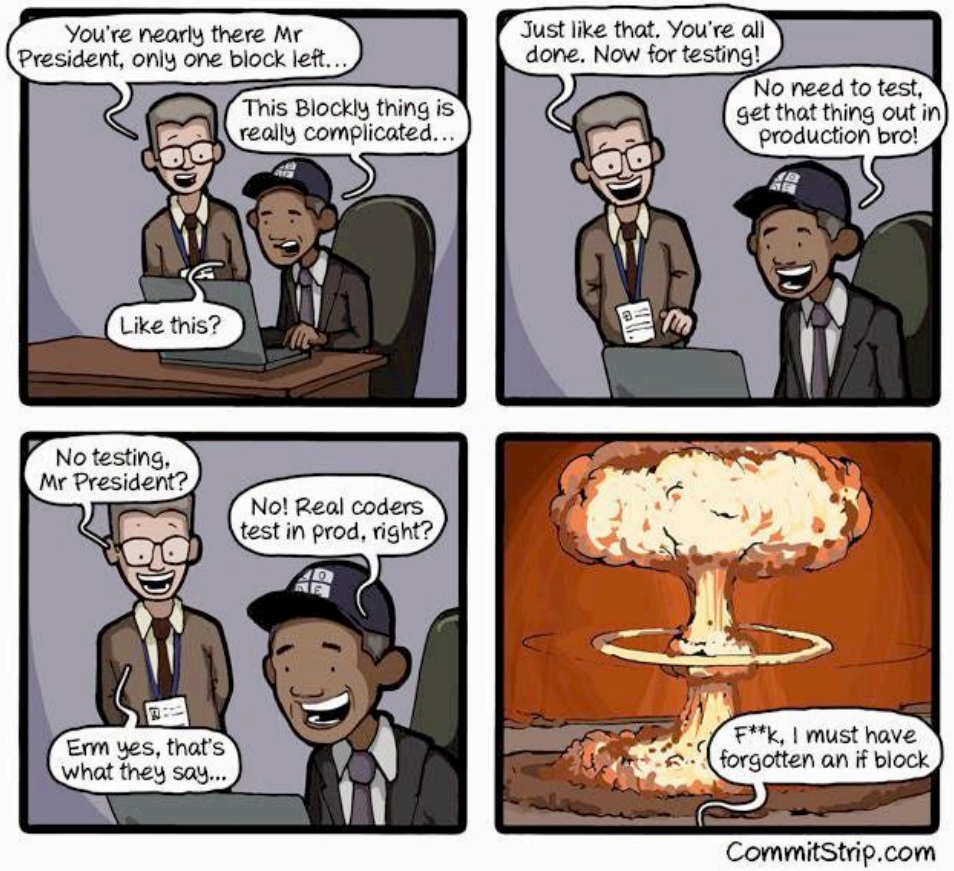
What to do in more complex scenarios?

„Yeah, bit I cannot unit test my code, because I use a [database, network, put your favourite external component here]!

Not really true, in these cases unit testing may become a bit more challenging, but is still possible and important!

- External components can be hidden behind interface classes
- Certain classes might only work in conjunction with others
- Some classes may need complex input data to execute their code

The python `unittest.mock` library can help to replace any python method called in your class/function by test code.



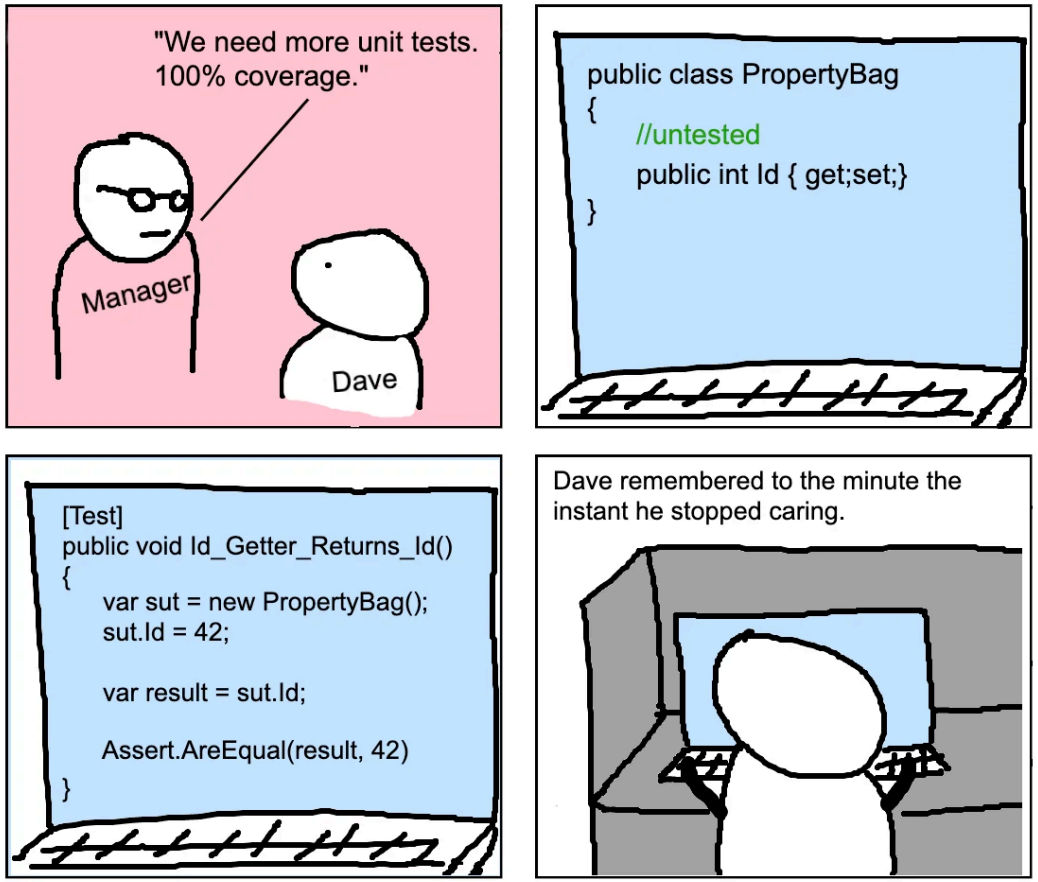
Test Coverage

The python coverage tool can analyze which part of the code base is actually executed aka covered by the unit test.

To execute and analyze the unit tests:

- To install the coverage package
  `python3 -m pip install coverage`
- To execute and analyze the unit tests
  `python3 -m coverage run ./my_tests.py`
- To create a text report
  `python3 -m coverage report`
- To create a html report
  `python3 -m coverage html`

The test coverage can be a quality criteria of a software project and a test coverage of 100% is desirable.

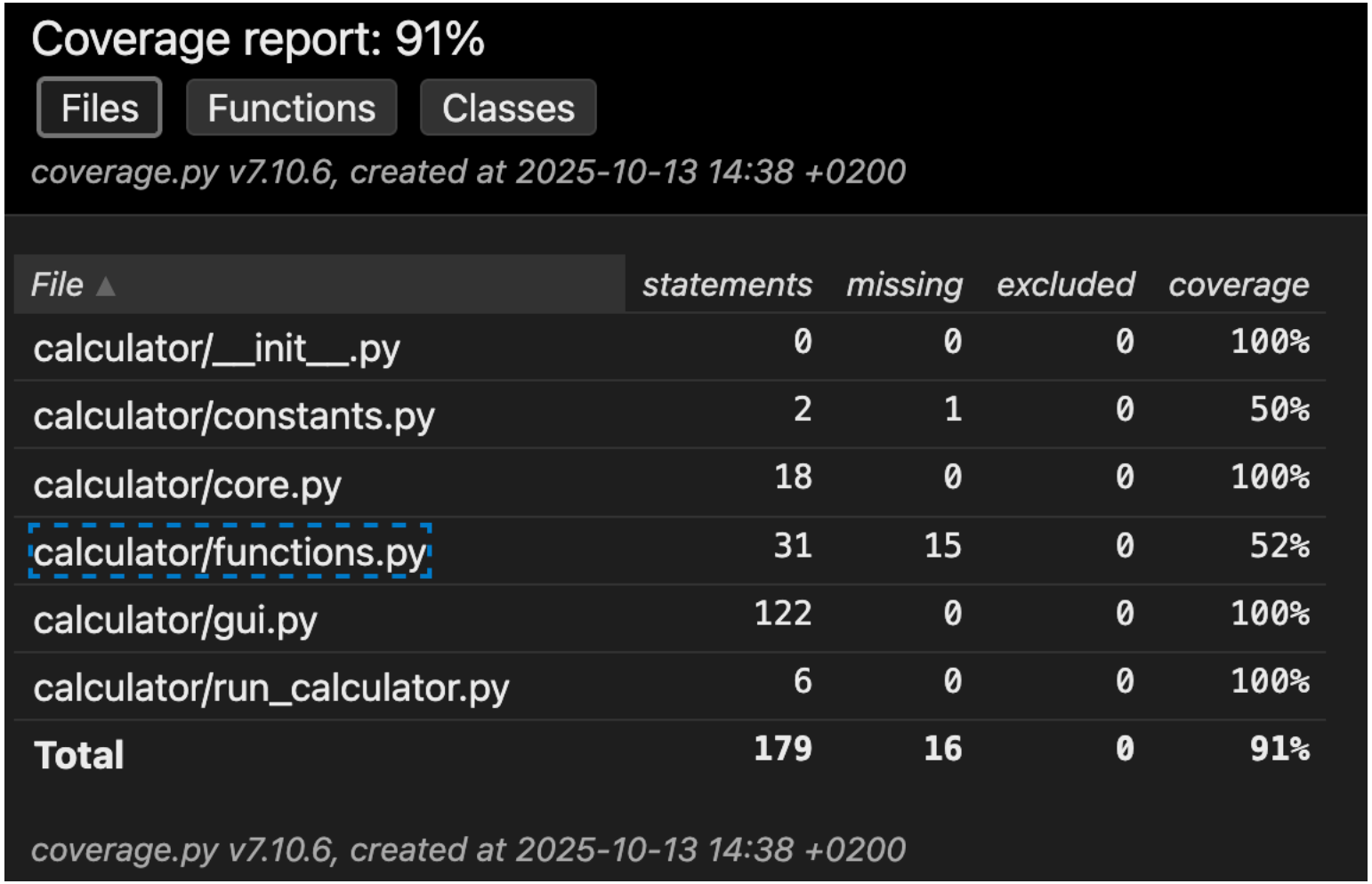


Coverage Report

```
giffler  MacBook-Pro2023  (e) venv  …  kseta-workshops  kseta-csd-2025  scientific-calculator  main  ./run_coverage.sh
.........................
----------------------------------------------------------------------
Ran 27 tests in 1.215s

OK
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
calculator/__init__.py            0      0   100%
calculator/constants.py           2      1    50%   2
calculator/core.py               18      0   100%
calculator/functions.py          31     15    52%   9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65
calculator/gui.py               122      0   100%
calculator/run_calculator.py      6      0   100%
-----------------------------------------------------------
TOTAL                           179     16    91%
```

Coverage Report



Coverage report: 91%

Files | Functions | Classes

coverage.py v7.10.6, created at 2025-10-13 14:38 +0200

File ▲	statements	missing	excluded	coverage
calculator/__init__.py	0	0	0	100%
calculator/constants.py	2	1	0	50%
calculator/core.py	18	0	0	100%
calculator/functions.py	31	15	0	52%
calculator/gui.py	122	0	0	100%
calculator/run_calculator.py	6	0	0	100%
Total	**179**	**16**	**0**	**91%**

coverage.py v7.10.6, created at 2025-10-13 14:38 +0200

Test Driven Development

Test Driven Development is a programming technique in which the requirement on a piece of code is expressed with a unit test, before the code is written

Typical workflow:

1. Write a unit test for the feature you are going to implement
2. Write the most minimalistic code to pass the unit test
3. Run unit test
4. Goto 1, if unit test succeeds and Goto 2, if unit test fails

Advantages of Test Driven Development:

- Developer has to think about the interfaces and behaviour beforehand
- Code is written to be test-able
- Writing unit tests is part of the regular development workflow

Test Driven Development

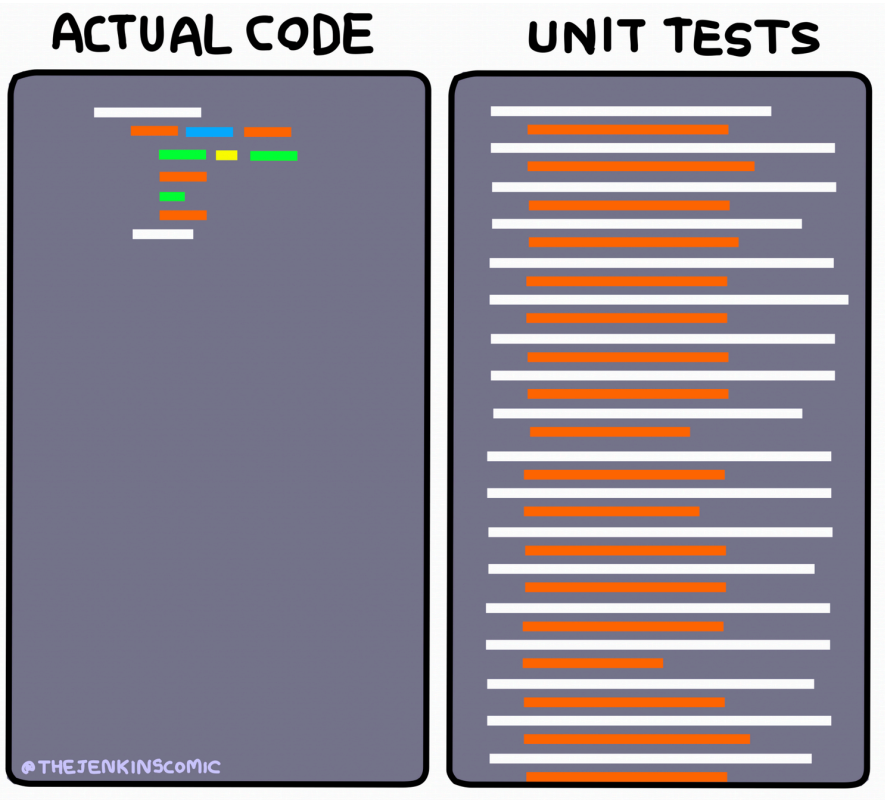
Test Driven Development is a programming technique in which the requirement on a piece of code is expressed with a unit test, before the code is written

Typical workflow:

1. Write a unit test for the feature you are going to implement
2. Write the most minimalistic code to pass the unit test
3. Run unit test
4. Goto 1, if unit test succeeds and Goto 2, if unit test fails

Advantages of Test Driven Development:

- Developer has to think about the interfaces and behaviour beforehand
- Code is written to be test-able
- Writing unit tests is part of the regular development workflow



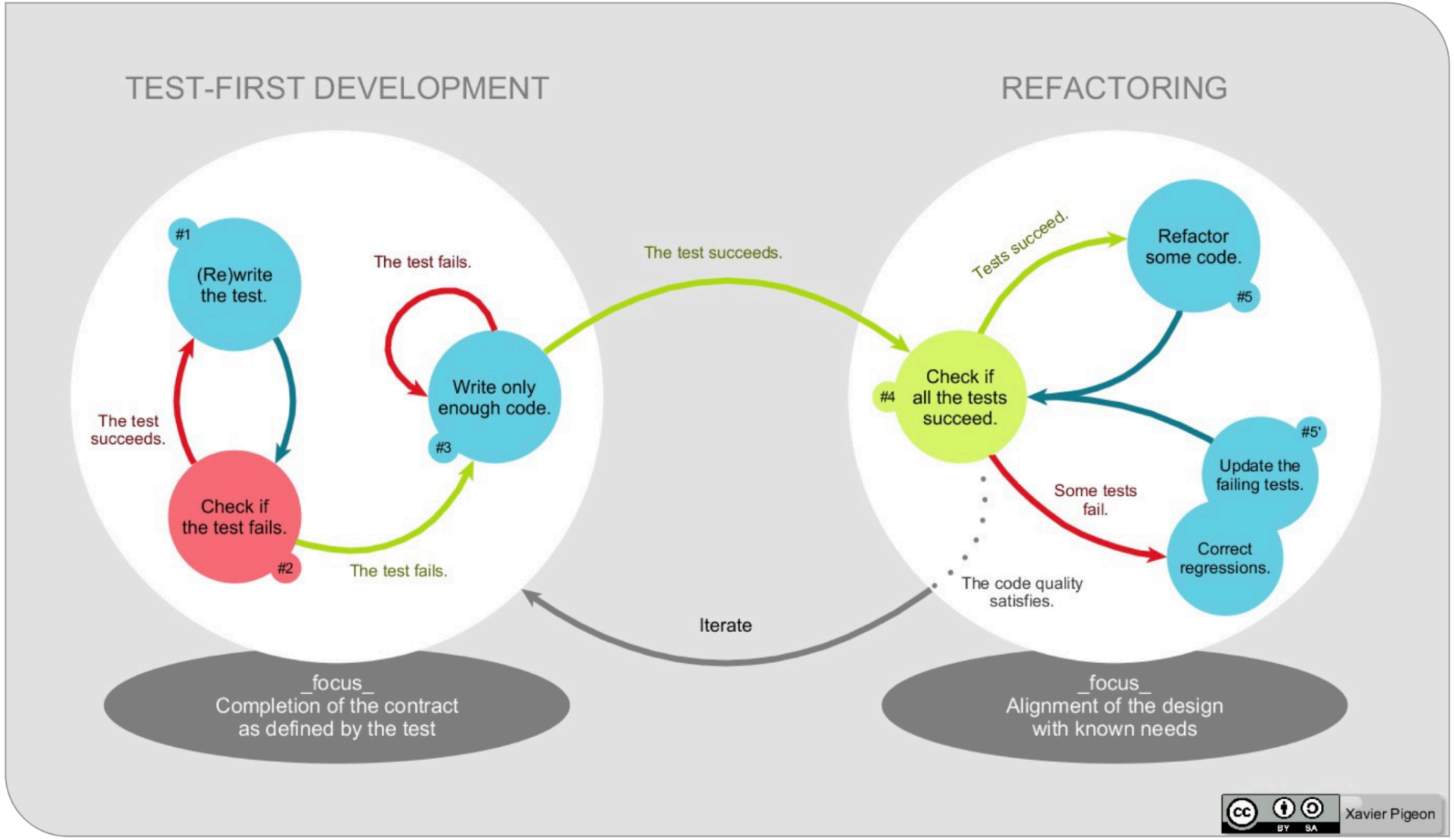

```
giffler MacBook-Pro2023 (e) venv … kseta-workshops kseta-csd-2025 scientific-calculator main cloc calculator
       5 text files.
       5 unique files.
       6 files ignored.

github.com/AlDanial/cloc v 2.06  T=0.01 s (504.5 files/s, 33098.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           5             63             23            242
-------------------------------------------------------------------------------
SUM:                             5             63             23            242
-------------------------------------------------------------------------------
giffler MacBook-Pro2023 (e) venv … kseta-workshops kseta-csd-2025 scientific-calculator main cloc tests
       5 text files.
       4 unique files.
       7 files ignored.

github.com/AlDanial/cloc v 2.06  T=0.01 s (421.1 files/s, 29160.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           4             49              6            222
-------------------------------------------------------------------------------
SUM:                             4             49              6            222
-------------------------------------------------------------------------------
```

Tests should be roughly the same order of magnitude in LOC as the code they test.

Test Driven Development



Questions?

Hands-on

Hands-on

- Please, form teams of two people for the next exercise
- Use Test-Driven Development (TDD) to implement as many missing features of the Scientific Calculator as possible, based on the user stories, in your private fork.

Important: Use one feature branch for one user story!