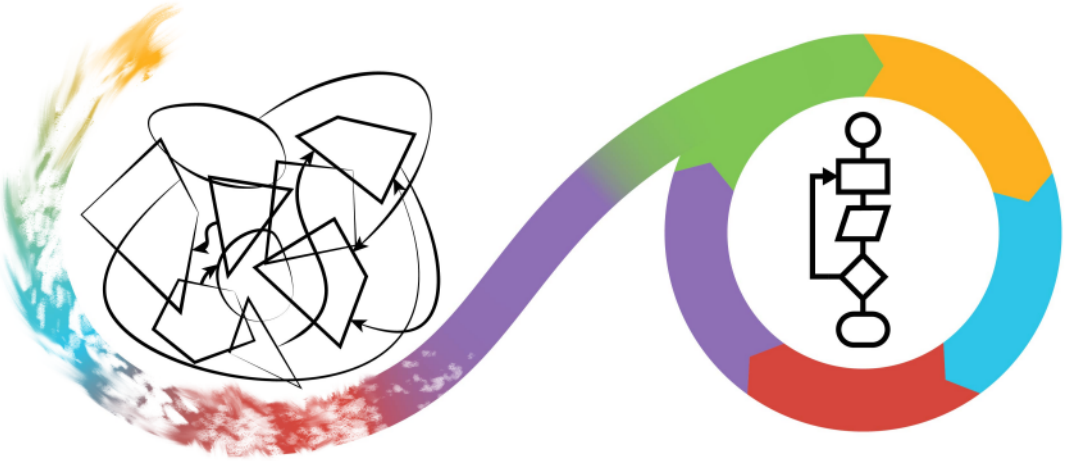
Testing

Richèl Bilderbeek

1 The Big Picture

https://github.com/UPPMAX/programming_formalisms/blob/main/tdd/tdd_lecture/tdd_lecture.qmd



1.1 Breaks

Please take breaks: these are important for learning. Ideally, do something boring (1)!

1.2 Schedule

Day	From	To	What
Wed	12:00	13:00	Lunch
Wed	13:00	13:45	Testing, `is_prime_[names]`
Wed	13:45	14:00	Break

Day	From	To	What
Wed	14:00	14:45	Testing, `flip_coin_[names]`
Wed	14:45	15:00	Break
Wed	15:00	15:30	Testing, `get_digits_[names]`
Wed	15:30	16:00	Reflection

2 Testing

2.1 Problems

When do you trust your code?

. . .

When do you trust code written by others?

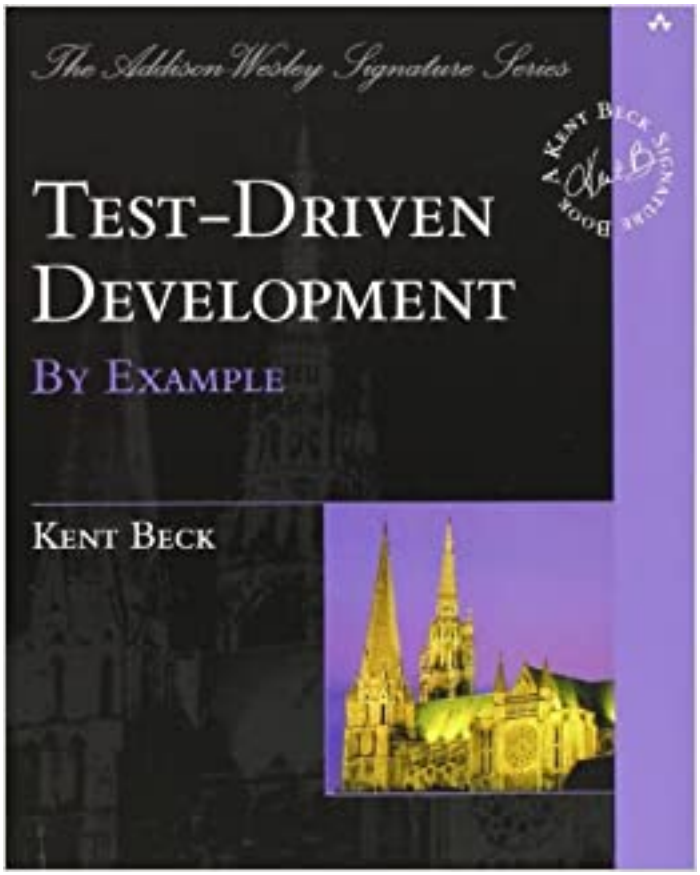
. . .

How do you convince other developers of a bug?

2.2 Testing

- Coding errors are extremely common (2)
- Contribute to the reproducibility crisis in science (3), e.g. (4)

Testing *helps* ensure the correctness of code.

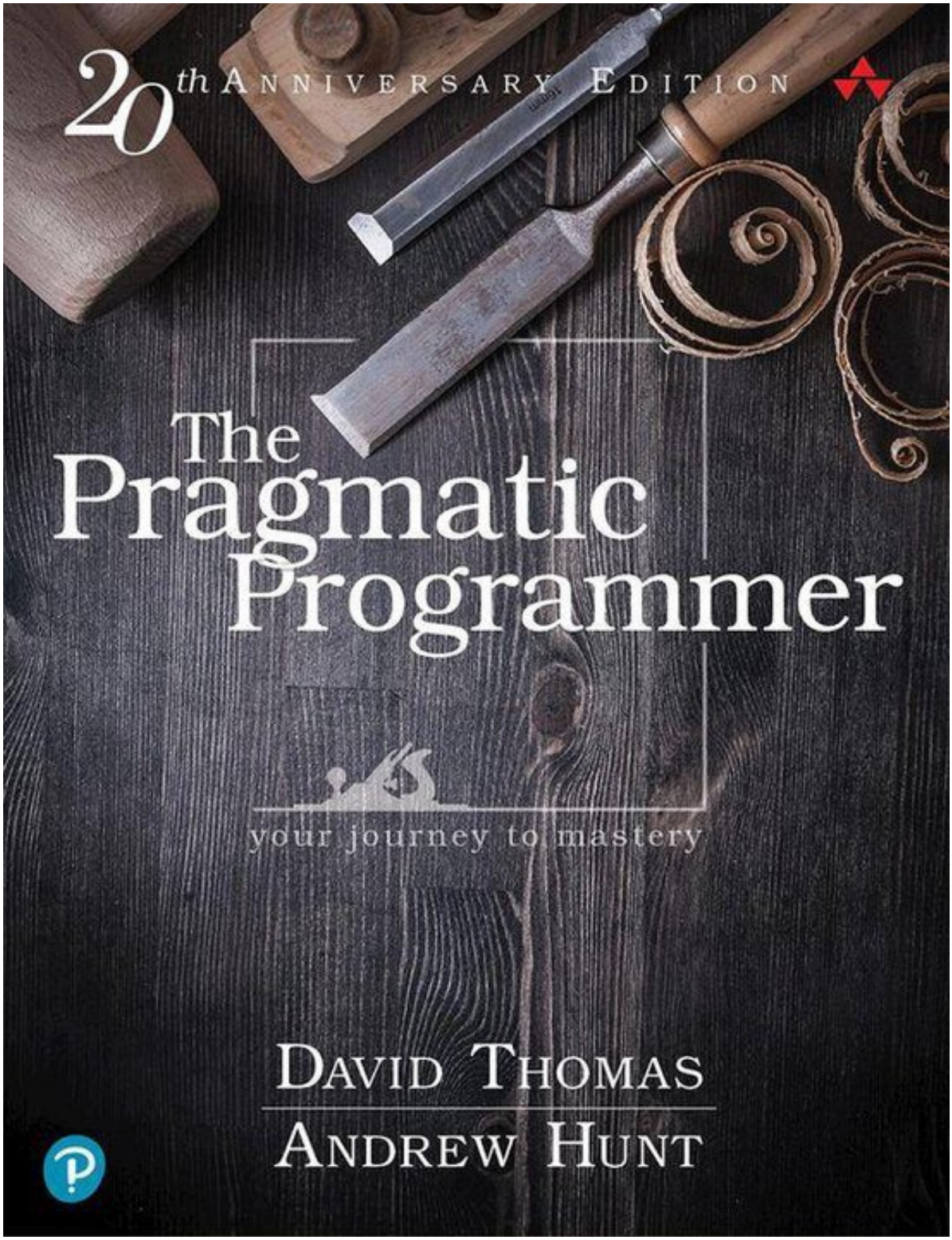


Modern C++ Programm
with Test-Driven Develo

Code Better,
Sleep Better

Jef

Edited by Mic


20th Anniversary Edition
The Pragmatic Programmer
your journey to mastery
David Thomas
Andrew Hunt

2.3 Testing framework

- `unittest`, `pytest`, `nose`, etc.
- Makes it easier to write unit tests
- Takes some scaffolding
- Failed tests give a better error message

2.4 Test if something is true

No testing framework:

```
assert 1 + 1 == 2
```

Using `unittest`:

```
import unittest

class TestSmall(unittest.TestCase):
    def test_is_true(self):
        self.assertIsTrue(1 + 1 == 2)
```

Mostly scaffolding here

2.5 Test if something is equal

No testing framework:

```
assert 1 + 1 == 2
```

Using `unittest`:

```
import unittest

class TestSmall(unittest.TestCase):
    def test_is_equal(self):
        self.assertEqual(1 + 1, 2)
```

Hamcrest notation can give better error message.

2.6 Test if something raises an exception

No testing framework:

```
def raise_error():
    raise RunType("Raise an error!")

has_raised = False
try:
    raise_error()
except:
    has_raised = True
assert has_raised
```

Using `unittest`:

```
import unittest

class TestSmall(unittest.TestCase):
    def test_raises(self):
        self.assertRaises(RunTimeError, raise_error)
```

here it pays off.

3 Example exercise: `is_prime`

- Only observe, no type-along!
- Ask questions on the go! When in doubt: ask that question!
- Time: 30 minutes

3.1 Exercise: `is_prime`

- Function name: `is_prime_[name]`, for example, `is_prime_richel`
- Output:
  - Returns `True` if the input is prime
  - Returns `False` if the input is not prime
  - Gives an error when the input is not an integer

3.2 Exercise: `is_prime`, social

- Ping-Pong Pair programming
- Discuss how and when to switch roles first!
- Person with first name first in alphabet starts
- Try to be **an exemplary duo**

3.3 Exercise: `is_prime`, technical

- Work within scaffolding of the learners project
  - Functions are in `src/[package_name]/testing.py`
  - Tests are in `tests/test_testing.py`
- Work on the main branch only, share code using `git push` and `git pull`
- order the `is_prime_[name]` functions and tests alphabetically, e.g. `is_prime_lars` comes before `is_prime_richel`

3.4 Live demo (30 minutes)

- Or videos: YouTube download (.ogv)

4 Exercise 1: `is_prime`

- Time: 30 minutes
- Do the same exercise in pairs
- There are multiple ways to do this: pick the way you feel is most natural

5 Continuous integration

5.1 Problem

How to work together well?

. . .

Encourage/enforce:

- Code must pass all tests
- High code coverage
- Uniform coding style

- URL links are valid
- Correct spelling

5.2 Continuous Integration

Scripts that are triggered when `push`ing code.

Assures quality:

- Tests pass
- Code has consistent style
- Links are valid (i.e. not broken)
- Spelling is correct
- [your check here]

5.3 Continuous Integration

- CI significantly increase the number of bugs exposed (5)
- CI increases the speed at which new features are added (5)

-> Demo on learners repo

5.4 Code coverage

- Percentage of code tested
- Correlates with code quality (6) (7)
- 100% mandatory to pass a code peer-review by rOpenSci (8)

5.5 Coding style

- Following a consistent coding style improves software quality (9)
  - Python: PEP8 (10)
  - R: Tidyverse (11)
- May include cyclomatic complexity
  - More complex code, more bugs (12) (13) (14)

5.6 Coding style tools

- Linter: program that tests code for style.

In Python: `ruff`, Sonar, `pytype`, Black, Codacy, Pylint, Flake8, `autopep8`, Pychecker, Pylama

5.7 Disable a `ruff` test

```
import random
i = random.randint(0, 1) # noqa: S311
```

You will need to defend this in a code review.

5.8 Testing indeterministic functions

Functions that do not always return the same values.

```
def flip_coin():
    """Produce a random boolean."""
    return random.randint(0, 1) > 0
```

How to test these?

5.9 Randomness

A Random Number Generator ('RNG') produces the same random values after setting the same RNG seed.

```
import random
random.seed(5)
assert flip_coin()
random.seed(2)
assert not flip_coin()
```

6 Exercise 2: `flip_coin`

- Time: 45 minutes

6.1 Exercise 2: `flip_coin`

- Function name: `flip_coin_[name]`, for example, `flip_coin_richel`
- Input: none
- Output: Returns `True` in 50% of all cases, else returns `False`
- **Get all CI scripts to pass**

6.2 Exercise 2: `flip_coin`, social

- Ping-Pong Pair programming
- Discuss how and when to switch roles first!
- Person with first name first in alphabet starts
- Try to be **an exemplary duo**

6.3 Exercise 2: `flip_coin`, technical

- Work within scaffolding of the learners project
  - Functions are in `src/[package_name]/testing.py`
  - Tests are in `tests/test_testing.py`
- Work on the main branch only, share code using `git push` and `git pull`
- order the `flip_coin_[name]` functions and tests alphabetically, e.g. `flip_coin_lars` comes before `flip_coin_richel`

7 Tests in a team

If all tests pass, we are -by definition- happy.

Programming team tresinformal

7.1 Problem

Q: When one works in a team, how to make sure my code keeps doing the same?

```
def get_test_dna_sequence():
  """Get a DNA sequence to be used in testing"""
  return "ACGTACGT"
```

. . .

A: Apply the Beyoncé Rule

7.2 Beyoncé rule

'If you like it, then you gotta put a test on it'

```
assert get_test_dna_sequence() == "ACGTACGT"
```

Teams should be reluctant to change tests: this will likely break other code.

Source: Wikimedia

7.3 Untestable functions

Q: How to test this function?

```
def print_hello():
    print("Hello world")
```

. . .

A: Never write untestable functions

7.4 Making untestable functions testable

Q: How to make this function testable?

```
def print_hello():
    print("Hello world")
```

. . .

```
def get_hello_world_text():
    return "Hello world"
```

7.5 Testing graphical functions

Q: How to test this function thoroughly:

- Plot looks pretty
- Colors are correct
- Trend line is drawn

Figure 1: Beyoncé

```
def save_plot(filename, x_y_data):
    """Save the X-Y data as a scatter plot"""
```

. . .

A: usually: use **a human**, e.g. a code reviewer

In most cases, graphical analysis tools and/or AI are overkill. If you are stubborn: try!

8 Exercise 3: `get_digits`

- Time: 45 minutes

8.1 Exercise 3: `get_digits`

- Function name: `get_digits_[name]`, for example, `get_digits_richel`
- Input: a positive number
- Output:
  - Returns the number split into a list of digits, e.g. `314` become `[3, 1, 4]`
  - Gives an error when the input is not a positive integer
- **Get all CI scripts to pass**

8.2 Exercise 3: `get_digits`, social

- Ping-Pong Pair programming
- Discuss how and when to switch roles first!
- Person with first name first in alphabet starts
- Try to be **an exemplary duo**

8.3 Exercise 3: `get_digits`, technical

- Work within scaffolding of the learners project
  - Functions are in `src/[package_name]/testing.py`
  - Tests are in `tests/test_testing.py`
- Work on the main branch only, share code using `git push` and `git pull`
- order the `get_digits_[name]` functions and tests alphabetically, e.g. `get_digits_lars` comes before `get_digits_richel`

8.4 Exercise 3: solution

`get_digits` video:

- download (.ogv)
- YouTube

9 Recap

- Testing helps code correctness
  - Use the Beyoncé Rule on precious behavior
- Testing + CI:
  - Helps teaching
  - Helps bug reporting

9.1 Weaknesses

- We developed only simple algorithms
- We only use simple data structures
- We ignore if code is fast [*]

This will be addressed in the next lectures :-)

- [*] vague wording on purpose

9.2 Questions?

Questions?

9.3 The End

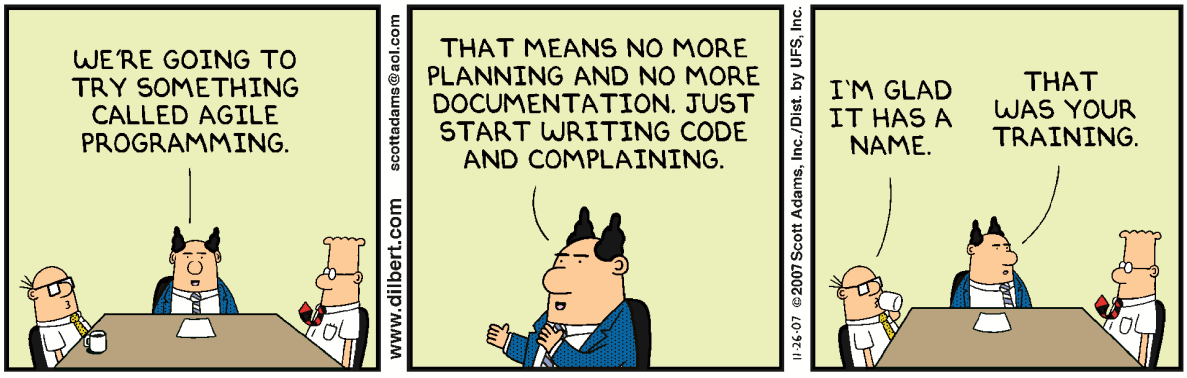
9.4 Links

- Former lecture on testing
- Hypermodern Python Cookiecutter
- Scikit-HEP project info for developers

10 Breaks

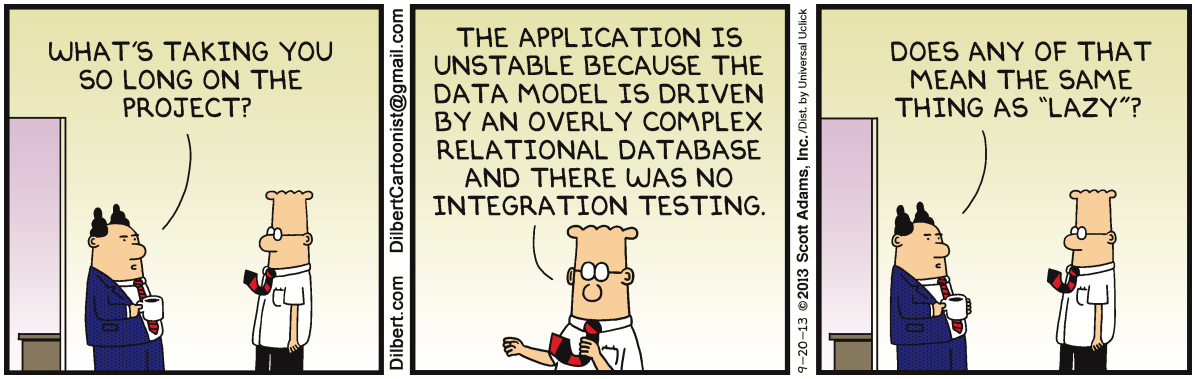
I put the break slides in the end

10.1 Break 1: 13:45-14:00



10.2 Break 2: 14:45-15:00

10.3 Done: 16:00



References

1. Newport C. Deep work: Rules for focused success in a distracted world. Hachette UK; 2016.
2. Baggerly KA, Coombes KR. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. The Annals of Applied Statistics. 2009;1309–34.
3. Vable AM, Diehl SF, Glymour MM. Code review as a simple trick to enhance reproducibility, accelerate learning, and improve the quality of your team's research. American Journal of Epidemiology. 2021;190(10):2172–7.
4. Rahman A, Farhana E. An exploratory characterization of bugs in covid-19 software projects. arXiv preprint arXiv:200600586. 2020;

5. Vasilescu B, Yu Y, Wang H, Devanbu P, Filkov V. Quality and productivity outcomes relating to continuous integration in GitHub. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering. ACM; 2015. p. 805–16.

6. Horgan JR, London S, Lyu MR. Achieving software quality with testing coverage measures. Computer. 1994;27(9):60–9.

7. Del Frate F, Garg P, Mathur AP, Pasquini A. On the correlation between code coverage and software reliability. In: Software reliability engineering, 1995 Proceedings, sixth international symposium on. IEEE; 1995. p. 124–32.

8. Ram K, Boettiger C, Chamberlain S, Ross N, Salmon M, Butland S. A community of practice around peer review for long-term research software sustainability. Computing in Science & Engineering. 2018;21(2):59–65.

9. Fang X. Using a coding standard to improve program quality. In: Quality software, 2001 Proceedings Second asia-pacific conference on. IEEE; 2001. p. 73–8.

10. Van Rossum G, Warsaw B, Coghlan N. PEP 8–style guide for Python code. Python org. 2001;1565.

11. Wickham H. Advanced R. CRC press; 2019.

12. Abd Jader MN, Mahmood RZ. Calculating McCabe's cyclomatic complexity metric and its effect on the quality aspects of software. 2018;

13. Chen C. An empirical investigation of correlation between code complexity and bugs. arXiv preprint arXiv:191201142. 2019;

14. Zimmermann T, Nagappan N, Zeller A. Predicting bugs from history. In: Software evolution. Springer; 2008. p. 69–88.