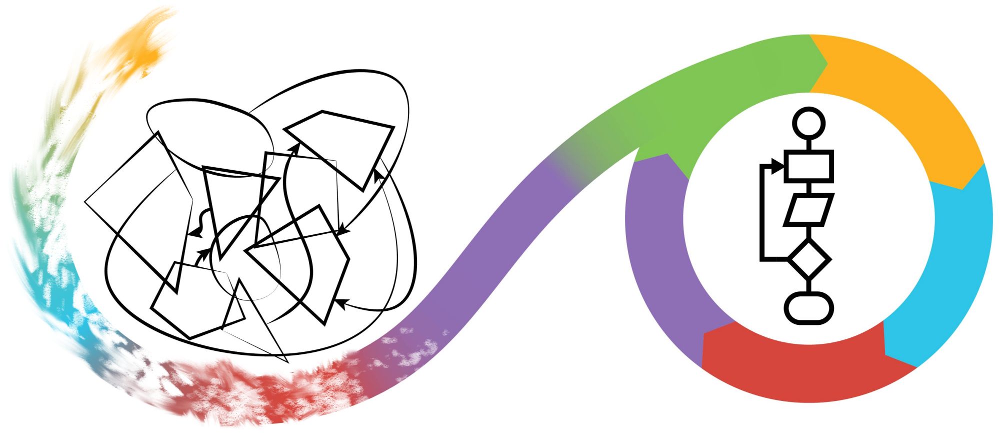
Testing

Richèl Bilderbeek

Testing

https://github.com/UPPMAX/programming_formalisms/blob/main/testing/testing_lecture/testing_lecture.qmd



Problems

When do you trust your code?

. . .

When do you trust code written by others?

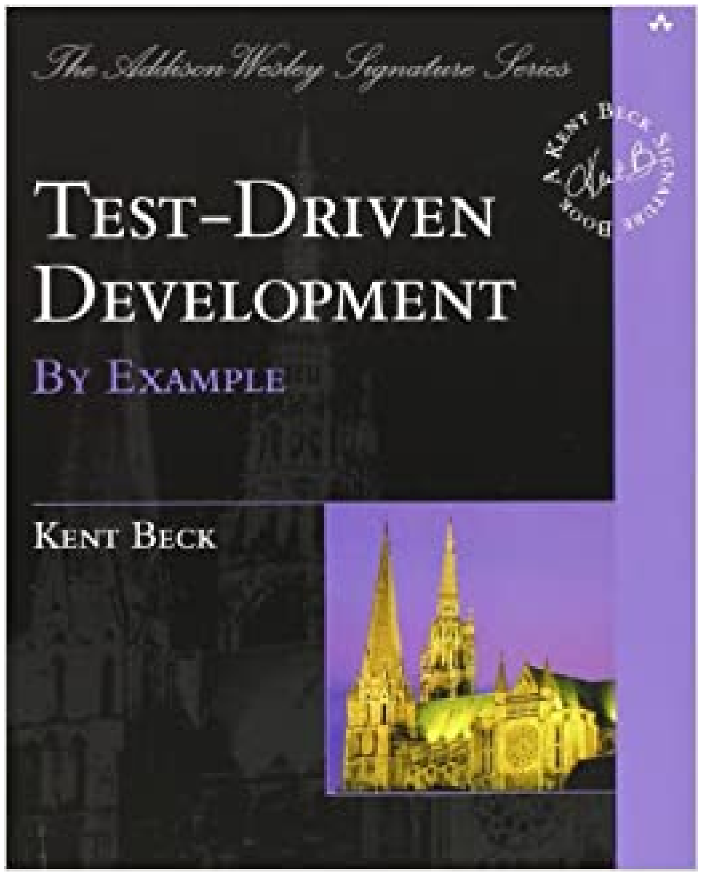
. . .

How do you convince other developers of a bug?

Testing

- Coding errors are extremely common (1)
- Contribute to the reproducibility crisis in science (2), e.g. (3)

Testing *helps* ensure the correctness of code.


The Addison Wesley Signature Series

A Kent Beck Signature Book

TEST-DRIVEN DEVELOPMENT

BY EXAMPLE

KENT BECK

Copyrighted Material

The Pragmatic Programmers

Modern C++ Programm

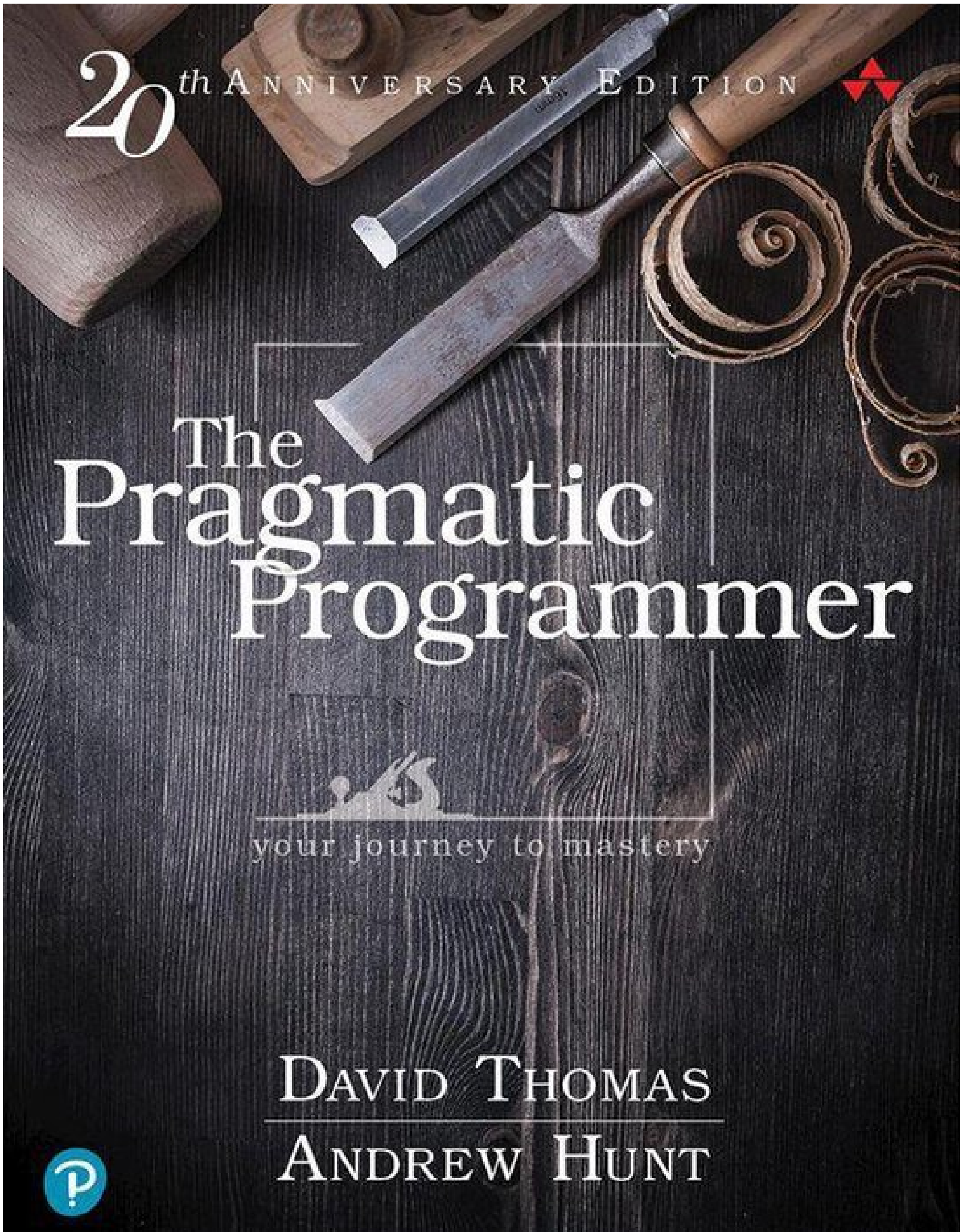
with Test-Driven Develo

Code Better, Sleep Better

Jef

Edited by Mic

Copyrighted Material


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

Testing framework

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

Test if something is true

No testing framework:

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

Using `unittest`:

```python
import unittest

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

Mostly scaffolding here

Test if something is equal

No testing framework:

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

Using `unittest`:

```python
import unittest

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

Hamcrest notation can give better error message.

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.

First example: `is_prime`

- Function name: `is_prime`
- 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
- Work within scaffolding of https://github.com/richelbilderbeek/programming_formalisms_testing

Live demo (15 minutes)

- Or videos: YouTube download (.ogv)

Exercise: `is_prime`, form

- Pair up
- Switch roles every 3 minutes
- Discuss how to keep the time first
- Person with GitHub username first in alphabet starts
- Work on `master` branch only, share code using `git push` and `git pull`
- Try to be **an exemplary duo**

Exercise: `is_prime`, technical

- Create a Fork of https://github.com/richelbilderbeek/programming_formalisms_testing
- Develop a function called `is_prime`:
  - `src/pftesting_richelbilderbeek/testing_questions.py` (has more questions)
  - `tests/test_testing_questions.py`
- Modify `README.md`: replace `richelbilderbeek` by your own username
- Extra: make all tests pass

Break 1

Break!

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

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]

Continuous Integration

- CI significantly increase the number of bugs exposed (4)
- CI increases the speed at which new features are added (4)

Code coverage

- Percentage of code tested
- Correlates with code quality (5) (6)
- 100% mandatory to pass a code peer-review by rOpenSci (7)

Coding style

- Following a consistent coding style improves software quality (8)
    - Python: PEP8 (9)
    - R: Tidyverse (10)
- May include cyclomatic complexity
    - More complex code, more bugs (11) (12) (13)

Coding style tools

- Linter: program that tests code for style.

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

Disable a `ruff` test

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

You will need to defend this in a code review.

Untestable functions

Q: How to test this function?

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

. . .

A: Never write untestable functions

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"
```

Testing graphical functions

Q: How to test this function thoroughly:

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

```
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!

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?

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()
```

Exercise: `flip_coin`, form

- Pair up again
- Switch roles every 3 minutes
- Discuss how to keep the time first
- Person with GitHub username first in alphabet starts
- Work on `master` branch only, share code using `git push` and `git pull`
- Try to be **an exemplary duo**

Exercise: `flip_coin`, technical

- Create a Fork of https://github.com/richelbilderbeek/programming_formalisms_testing
- Modify `README.md`: replace `richelbilderbeek` by your own username
- Develop a function called `flip_coin`:
  - `src/pftesting_richelbilderbeek/testing_questions.py` (has more questions)
  - `tests/test_testing_questions.py`
- **Get all CI scripts to pass**

Break 2

> If all tests pass, we are -by definition- happy.
>
> Programming team tresinformal

Break 2

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

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

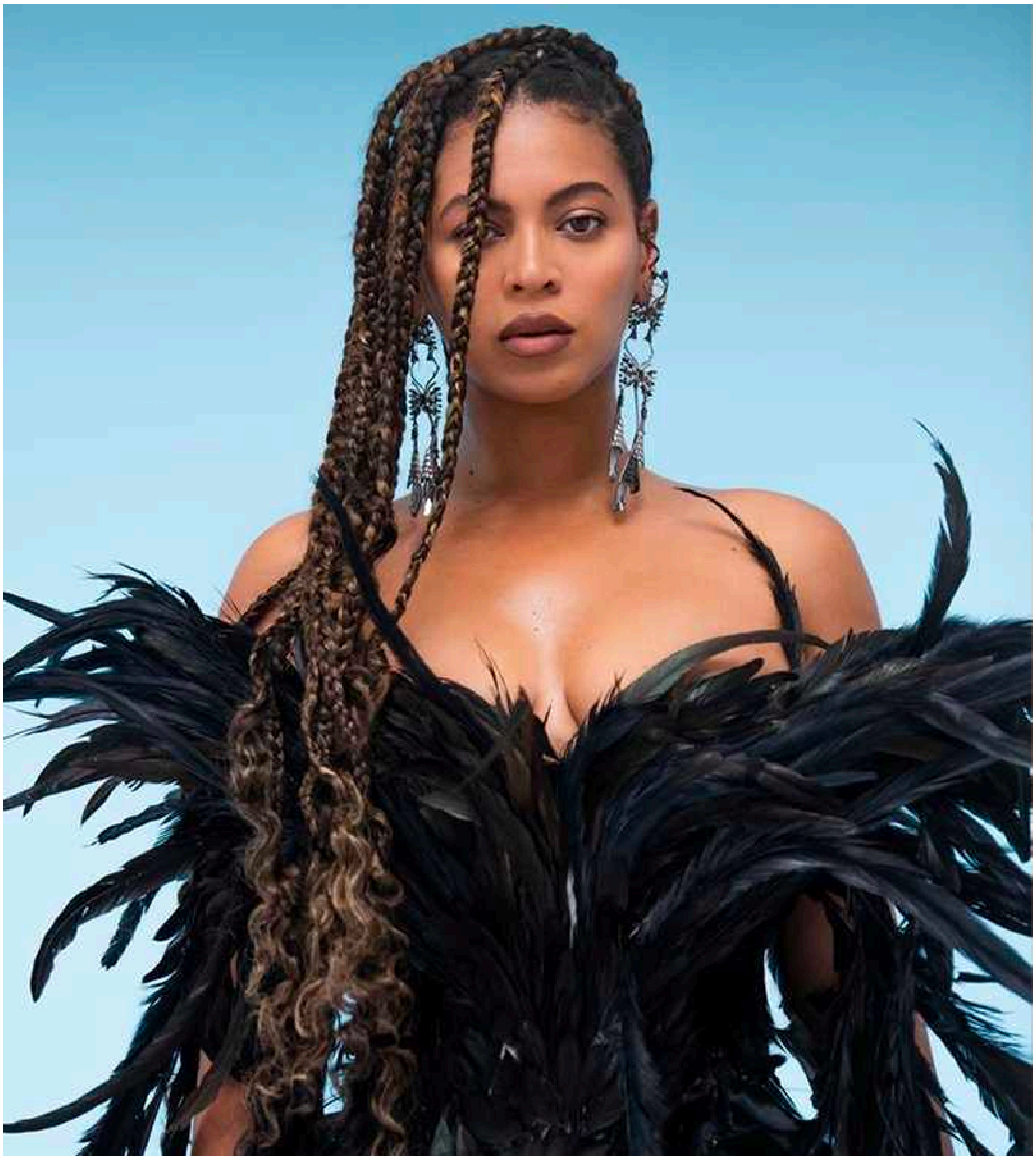


Figure 1: Beyoncé

Problem

Q: how to counter 'bit rot' (14) or 'software collapse' (15)

(we know: impossible to counteract (16))

. . .

A:

- Xtreme programming: 'embrace change' (17)
- Add a scheduled monthly test on your CI script

Consider static type checking

- Use static type checking, see PEP 484

```
def reverse_string(s: str) -\> str: return s.reverse()
```

Exercise: `get_digits`, form

- Pair up again
- Switch roles every 3 minutes
- Discuss how to keep the time first
- Person with GitHub username first in alphabet starts
- Work on `master` branch only, share code using `git push` and `git pull`
- Try to be **an exemplary duo**

Exercise: `get_digits`, technical

- Create a Fork of https://github.com/richelbilderbeek/programming_formalisms_testing
- Modify `README.md`: replace `richelbilderbeek` by your own username
- Develop a function called `get_digits`:
  - `src/pftesting_richelbilderbeek/testing_questions.py` (has more questions)
  - `tests/test_testing_questions.py`
- **Get all CI scripts to pass**

Solutions

`get_digits` video:

- download (.ogv)
- YouTube

Recap

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

Problems

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

Finally

Time for a Reflection!

Afterwards, you can rest or ask your final questions.

The End

The End

Links

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

1. 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.
2. 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.
3. Rahman A, Farhana E. An exploratory characterization of bugs in covid-19 software projects. arXiv preprint arXiv:200600586. 2020;
4. 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.
5. Horgan JR, London S, Lyu MR. Achieving software quality with testing coverage measures. Computer. 1994;27(9):60–9.
6. 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.
7. 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.
8. 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.
9. Van Rossum G, Warsaw B, Coghlan N. PEP 8–style guide for Python code. Python org. 2001;1565.
10. Wickham H. Advanced R. CRC press; 2019.
11. Abd Jader MN, Mahmood RZ. Calculating McCabe's cyclomatic complexity metric and its effect on the quality aspects of software. 2018;
12. Chen C. An empirical investigation of correlation between code complexity and bugs. arXiv preprint arXiv:191201142. 2019;
13. Zimmermann T, Nagappan N, Zeller A. Predicting bugs from history. In: Software evolution. Springer; 2008. p. 69–88.
14. Steele Jr GL, Woods DR, Finkel RR, Stallman RM, Goodfellow GS. The hacker's dictionary: A guide to the world of computer wizards. Harper & Row Publishers, Inc.; 1983.

15. Hinsen K. Dealing with software collapse. Computing in Science & Engineering. 2019;21(3):104–8.

16. Benureau FC, Rougier NP. Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. Frontiers in neuroinformatics. 2018;11:69.

17. Beck K. Extreme programming explained: Embrace change. Addison-Wesley Professional; 2000.