AUTOMATED TESTING WITH PYTEST

OR

MAKING PYTEST YOUR BATMAN 

WHO AM I?

Josh Kelley

WHAT IS AUTOMATED TESTING?

WRITE SOME CODE

Fibonacci sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ...

0 + 1 = 1

1 + 1 = 2

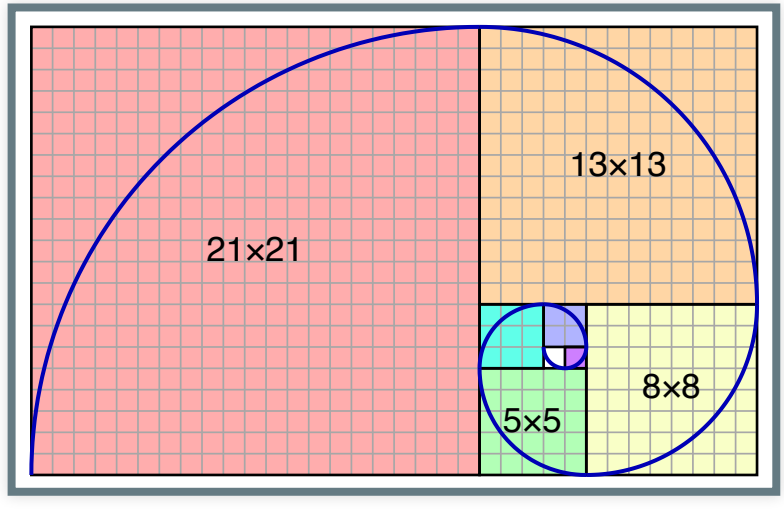
1 + 2 = 3

2 + 3 = 5

WRITE SOME CODE

Fibonacci sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ...



WRITE SOME CODE

Fibonacci sequence:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

TEST IT OUT

```
In [1]: from fib import fib

In [2]: fib(1)
Out[2]: 1

In [3]: fib(3)
Out[3]: 2

In [4]: fib(4)
Out[4]: 3

In [5]: fib(5)
Out[5]: 5
```

TEST IT OUT

```
In [6]: assert fib(2) == 1

In [7]: assert fib(5) == 5

In [8]: assert fib(6) == 8
```

TURN IT INTO SOMETHING YOU CAN RUN

```
In [9]: %save test_fib.py 1 6-8
```

```
# I can now re-run these tests whenever I want!
./test_fib.py
```

A TYPICAL SMALL AUTOMATED TEST (UNIT TEST)

A function or class with the following structure:

1. Arrange — set up inputs and preconditions
2. Act — do the operation
3. Assert — verify the results

```
employee = Employee('John', 'Doe')
employee.state = 'TN'
employee.city = 'Nashville'
employee.salary = 56516

payroll = employee.calculate_payroll()

assert payroll.withheld == 5000
```

WHAT'S THE BEST WAY TO DO AUTOMATED TESTING?

TERMINOLOGY

Automated testing
The opposite of manual testing

Unit testing
Testing a single function, class, or method

Test-driven development
Writing tests before you write the code

Integration testing or component testing
Testing a group of related units together

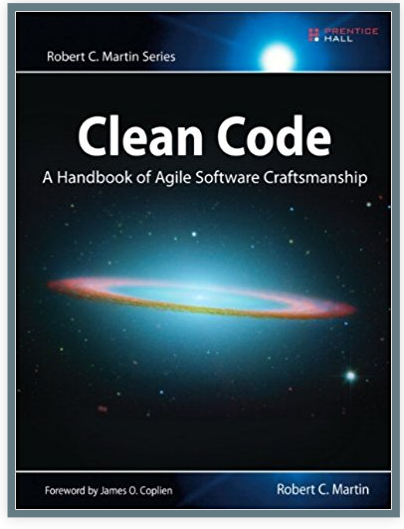
System testing or end-to-end testing
Tests that are run against the entire system

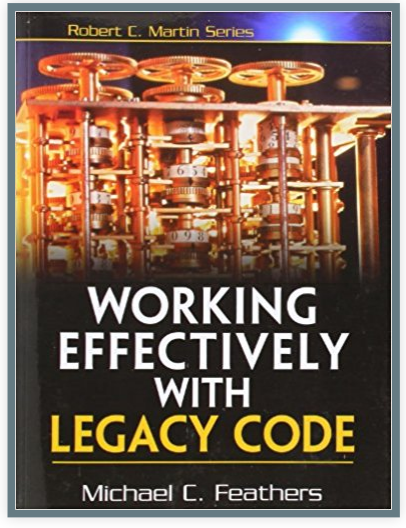
The Three Laws of TDD

(Test Driven Development)

1. You may not write production code until you have written a failing unit test.
2. You may not write more of a unit test than is sufficient to fail…
3. You may not write more production code than is sufficient to pass the currently failing test.

— Robert C. Martin, *Clean Code*



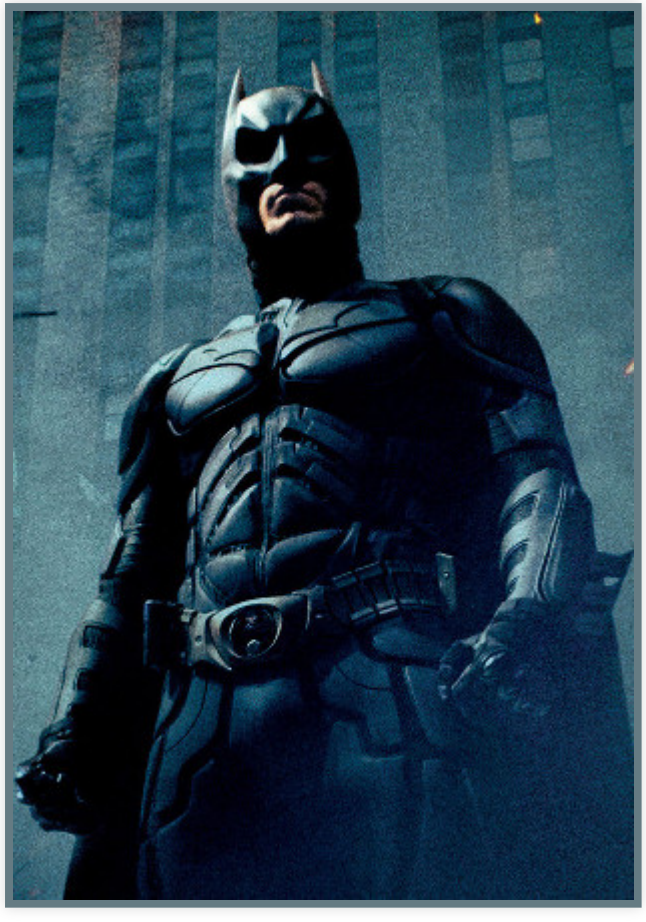


Unit tests run fast. If they don't run fast, they aren't unit tests.

Other kinds of tests often masquerade as unit tests. A test is not a unit test if:

1. It talks to a database.
2. It communicates across a network.
3. It touches the file system.
4. You have to do special things to your environment (such as editing configuration files) to run it.

— Michael Feathers, *Working Effectively with Legacy Code*



[I'm] lazy, as all good engineers are wont to be. Being lazy means you don't want to find bugs at run time. Being lazy means you don't want to ever make the same mistake twice. Being lazy means making your compiler(s) work as hard as possible, so that you don't have to... If you treat it well, you can make the compiler your right-hand man, helper, conscience, your batman.

— Matthew Wilson, *Imperfect C++*

[I'm] lazy, as all good engineers are wont to be. Being lazy means you don't want to find bugs at run time. Being lazy means you don't want to ever make the same mistake twice. Being lazy means making your compiler(s) work as hard as possible, so that you don't have to... If you treat it well, you can make the compiler your right-hand man, helper, conscience, your batman.

— Matthew Wilson, *Imperfect C++*

FASTER FEEDBACK

All the ways our industry has improved our feedback…

- Interactive computing
- Syntax highlighting
- IDE code analysis
- Agile software development
- Continuous integration

And automated testing

PUTTING IT INTO PRACTICE

UNITTEST

```
from fib import fib
import unittest

class FibTestCase(unittest.TestCase):
    def test_fib(self):
        self.assertEqual(fib(1), 1)
        self.assertEqual(fib(2), 1)
        self.assertEqual(fib(3), 2)
        self.assertEqual(fib(4), 3)
        self.assertEqual(fib(5), 5)

if __name__ == '__main__':
    unittest.main()
```

PYTEST BENEFITS

- Test discovery
- No need to inherit from TestCase
- Simpler assertions
- Plugins
- Test fixtures
- And lots of “quality of life” improvements

SWITCHING FROM UNITTEST

```
from fib import fib
import unittest

class FibTestCase(unittest.TestCase):
    def test_fib(self):
        self.assertEqual(fib(1), 1)
        self.assertEqual(fib(2), 1)
        self.assertEqual(fib(3), 2)
        self.assertEqual(fib(4), 3)
        self.assertEqual(fib(5), 5)

if __name__ == '__main__':
    unittest.main()
```

SWITCHING TO PYTEST

```
from fib import fib

def test_fib():
    assert fib(1) == 1
    assert fib(2) == 1
    assert fib(3) == 2
    assert fib(4) == 3
    assert fib(5) == 5
```

SWITCHING TO PYTEST

1. `pip install pytest`
2. Run `pytest` instead of `./my_test_suite.py`
3. That's it!

(or use unittest2pytest)

TEST DISCOVERY

By default:

- Search recursively under the current directory
- Any file named `test_*.py` or `*_test.py`
- Any class starting with `Test`
- Any function starting with `test_`

UNITTEST ASSERTIONS

```
assertEqual                    assertIsInstance
assertNotEqual                 assertIsNotInstance
assertTrue                     assertAlmostEqual
assertFalse                    assertNotAlmostEqual
assertIs                       assertGreater
assertIsNot                    assertGreaterEqual
assertIsNone                   assertLess
assertIsNotNone                assertLessEqual
assertIn                       assertRegex
assertNotIn                    assertNotRegex
```

PYTEST ASSERTIONS

```
assert
```

PYTEST ASSERTIONS

pytest inspects the Abstract Syntax Tree (AST) to turn this…

```
a = 1
b = 2
assert a + b == 3
```

PYTEST ASSERTIONS

...into this.

```
@py_assert0 = 1
@py_assert2 = 2
@py_assert4 = @py_assert0 + @py_assert2
@py_assert6 = 3
@py_assert5 = @py_assert4 == @py_assert6
if not @py_assert5:
    @py_format8 = @pytest_ar._call_reprcompare(('==',), \
        (@py_assert5,), ('(%(py1)s + %(py3)s) == %(py7)s',), \
        (@py_assert4, @py_assert6)) % {
            'py3': @pytest_ar._saferepr(@py_assert2),
            'py1': @pytest_ar._saferepr(@py_assert0),
            'py7': @pytest_ar._saferepr(@py_assert6)}
    @py_format10 = ('' + 'assert %(py9)s') % {'py9': @py_format8}
    raise AssertionError(@pytest_ar._format_explanation(@py_forma
@py_assert0 = @py_assert2 = @py_assert4 = @py_assert5 = @py_asser
```

(You are not expected to understand this.)

PYTEST ASSERTIONS

```
a = 1
b = 2
assert a + b == 4
```

Sample output:

```
        a = 1
        b = 2
>       assert a + b == 4
E       assert (1 + 2) == 4
```

PYTEST ASSERTIONS

```
actual   = {'first_name': 'John', 'last_name': 'Doe',
            'city': 'Nashville', 'state': 'TN'}

expected = {'first_name': 'John', 'last_name': 'Doe',
            'city': 'Memphis', 'state': 'TN'}

assert actual == expected
```

Sample output:

```
>       assert actual == expected
E       AssertionError: assert {'city': 'Nas...'state': 'TN'} ==
E         Omitting 3 identical items, use -vv to show
E         Differing items:
E         {'city': 'Nashville'} != {'city': 'Memphis'}
E         Use -v to get the full diff
```

PYTEST PLUGINS

docs.pytest.org/en/latest/plugins.html
plugincompat.herokuapp.com

PYTEST COVERAGE

```
pip install pytest-cov
pytest --cov=. --cov-report=html:../cov_html
```

PYTEST-WATCH

Run `ptw`, and your tests will be automatically executed whenever your code changes.

PYTEST-XDIST

Run tests across multiple CPUs or hosts.

Run tests repeatedly on file change (similar to pytest-watch).

Run tests on multiple environments.

USING PYTEST WITH PDB

Optionally, install PDB++ or pudb:

```
pip install pdbpp
```

Manually set a breakpoint:

```
import pdb; pdb.set_trace()
breakpoint()  # Python 3.7 only
```

Or break on the first failure:

```
pytest --pdb -x
```

EXAMPLE: REPORTING SLOW TESTS

Report the 3 slowest tests, so that you know what to work on:

```
pytest --durations=3
```

FILTERING TESTS

```
pytest test_notepad.py
pytest test_notepad.py::TestFileIO::test_save
```

EXAMPLE: EXPECTED ASSERTIONS

```
def test_negative_numbers():
    with pytest.raises(ValueError):
        fib(-1)
```

EXAMPLE: TEST FIXTURES

```
import pytest

@pytest.fixture
def smtp_connection():
    import smtplib
    return smtplib.SMTP("smtp.gmail.com", 587, timeout=5)

def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
```

EXAMPLE: MOCKS

```python
import my_api_client
import pytest
import requests_mock

@pytest.fixture
def m():
    with requests_mock.Mocker() as m:
        yield m

def test_api_client(m):
    # Tell the mock to expect a request for www.example.com
    # and send a reply of "Hello, world!" back.
    m.get('http://www.example.com', content='Hello, world!')

    # Execute the code under test.
    response = my_api_client.query()

    # Verify that our mock object was called.
    assert m.called
    assert m.call_count == 1

    assert response == 'Hello, world!'
```

EXAMPLE: MOCKS

- unittest.mock
- pytest's monkeypatch
- requests-mock
- FreezeGun

EVEN MORE FEATURES

- Test fixtures for managing temporary directories
- Capture log output
- Intercept Python warnings and deprecations

CASE STUDY: AUTOMATING GUI TESTS WITH PYWINAUTO

PYTEST FIXTURES

```
@pytest.fixture
def notepad():
    notepad = pywinauto.Application().start('notepad.exe')
    yield notepad
    notepad.close()

def test_copy(notepad):
    notepad.UntitledNotepad.Edit.type_keys('Hello')
    notepad.UntitledNotepad.Edit.type_keys('^A^C')
    assert pywinauto.clipboard.GetData() == 'Hello'
```

FURTHER READING

"Professionalism and Test-Driven Development," Robert C. Martin, IEEE Software, Vol. 24, Issue 3

UnitTest and SelfTestingCode, Martin Fowler

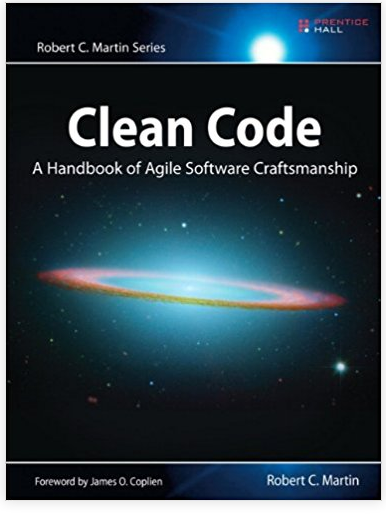
Why Most Unit Testing is Waste, James O. Copelien

TDD is dead. Long live testing, David Heinemeier Hansson

Just Say No to More End-to-End Tests, Mike Wacker

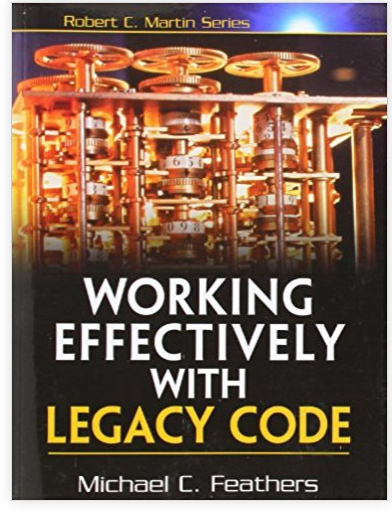
Write tests. Not too many. Mostly integration, Kent C. Dodds

FURTHER READING

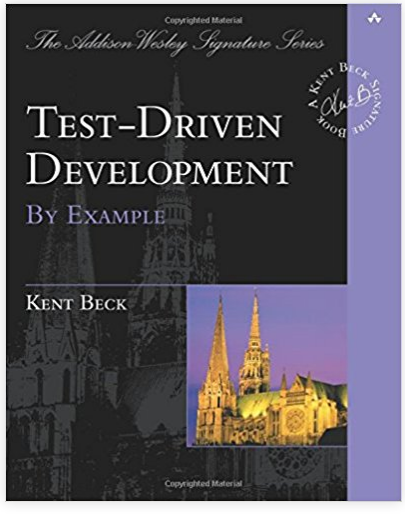


Clean Code, by Robert C. Martin

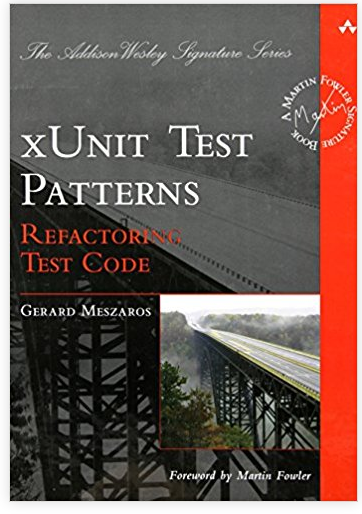
Working Effectively With Legacy Code, Michael Feathers



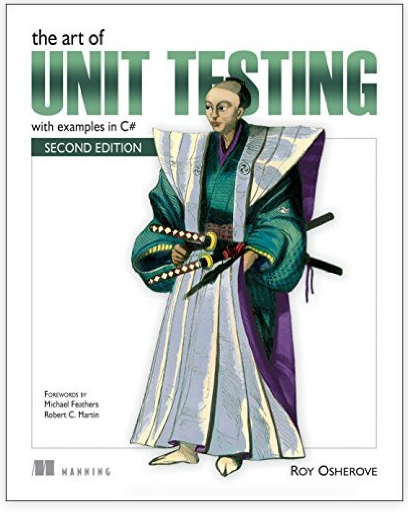
FURTHER READING



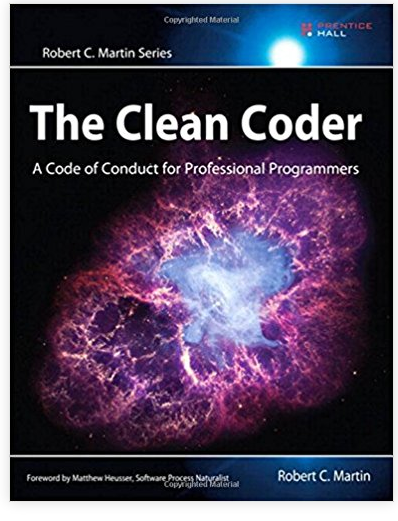
Test Driven Development: By Example, Kent Beck



xUnit Test Patterns, by Gerard Meszaros



The Art of Unit Testing, by Roy Osherove



The Clean Coder, by Robert C. Martin

IMAGE CREDITS

"Fibonacci Number," Wikipedia

The Dark Knight © Warner Bros. 2017.

The Lord of the Rings film trilogy © New Line Cinema 2001, 2002, 2003

 github.com/joshkel/automated-testing-with-pytest

 joshkel@gmail.com

 @notthatjoshkel