Object oriented programming

Unit testing

TDD (test driven development)

Why test?

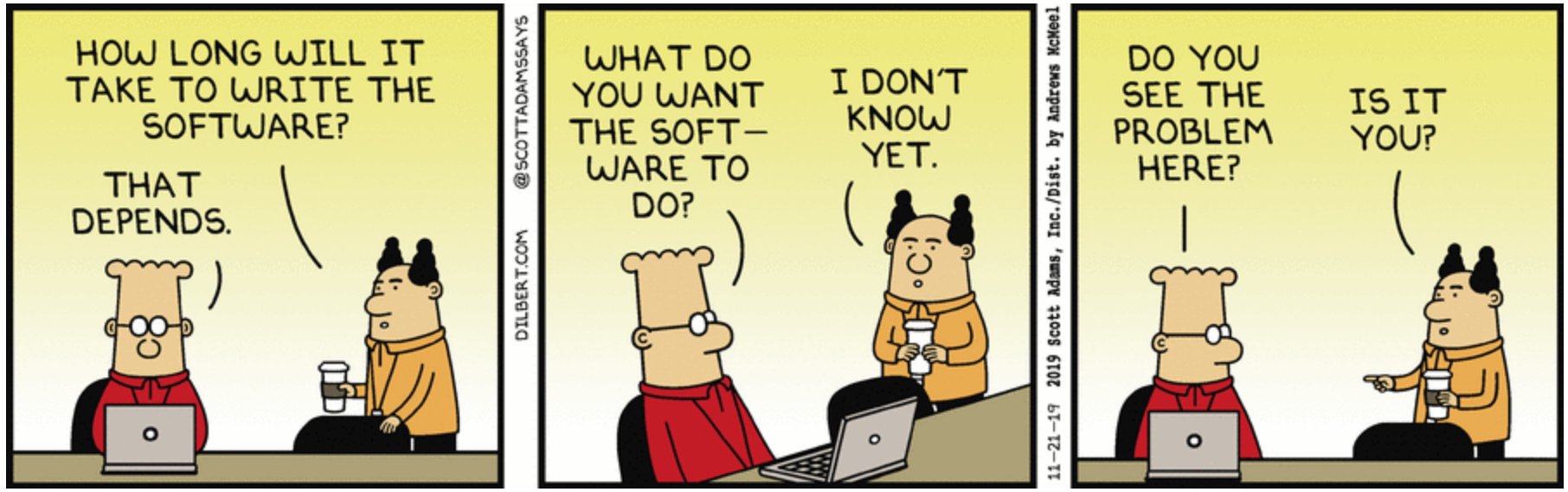
- You have been testing your code since day 1
- The only way to make sure "everything works"
- Automated tests are easier and quicker to run
- One button approach: you run a command, and you know whether or not your code works
- Unit tests are automated tests
- Unit tests operate at the lowest level possible: only validate the basic elements of a program
- Unit tests do not validate how objects interact with each other (this is *integration testing*)


We received your delivery on time, but we couldn't find the tests
Tests ?
What tests?
You know, the unit tests. The ones for testing the application layer, Quality Assurance and all
Ah, yes, the unit tests...
There aren't any unit tests
No tests ? But what about the guidelines...
How do you check that everything works correctly?
Um...
IT'S MAGIC
CommitStrip.com

Unit tests

- Are fast to run (a few seconds, maximum)
- Are easy to write (when written at the same time as the actual code)
- Can be automated and scheduled
  - Every night
  - Upon each commit in a git repository: continuous integration
- Unit tests can also help developers
  - reading the tests tell a lot about the public interface, and its model
- There is a development method where you write tests before / at the same time as writing the actual program. This is called **Test Driven Development** (TDD)
- This allows to make sure that the requirements for the programs are respected by the developer: the program does what it is supposed to do

Untested code is broken code



Testing in Python

- The testing framework provided by Python is called `unittest`
- It is derived from `jUnit` and has a very Java-oriented syntax
- It is good to know about `unittest`, but `pytest` is more convenient to use

Install pytest

- `pytest` is a Python package
- Install it in your **virtual environment**: `pip install pytest`

Run tests with `pytest`

- At the root of your project, run: `pytest`
- `pytest` will scan and discover all your tests in the test files
- Test files are any files that match the patterns:
  - `test_*.py`
  - `*_test.py`
  - these files can be in subdirectories / folders
- You can also run it with just the command `pytest`
- Tests are functions/methods whose names start with `test_`:
- Tests can be:
  - regular functions outside of classes
  - methods inside classes, whose names start with `Test`

> Very often, tests are located in the `tests` folder, at the root of the folder.

Examples

```
# test_something.py

def test_thingy():          # This is run by pytest
  pass

def thing():                # But not this
  pass

class Something:
  def test_thing(self):     # Also not this!
    pass

class TestSomething:
  def test_thing2(self):    # This is run by pytest
    pass
```

How to write a test

The code to be tested

```
def add_values(a, b):
    return a+b
```

The test

```
def test_add_values():
    result = add_values(2, 3)
    assert result == 5          # This line makes sure the output is the one we expect
```

We use `assert` to test the logic of our program.

How to test for exceptions?

- It is sometimes *expected* that your code raises exceptions

```
def add_values(a, b):
    if type(a) is not int or type(b) is not int:
        raise TypeError("Invalid value")
    return a+b
```

The test

```
import pytest

def test_add_values_invalid():
  with pytest.raises(TypeError):
    result = add_values([1], [2])
```

Code coverage and unit testing

- The goal of unit testing is to make sure every aspect of the code is tested and working
- We can measure "how much code" is covered by unit tests: the **code coverage**
- It is typical for teams to aim for at least 80% of code coverage, and sometimes even 90%
- Reference: 5 questions every unit test must answer

Code coverage in Python

- We use the `coverage` library (an external dependency), together with `pytest`
- You can install it in a virtual environment with `pip install pytest-cov`
- You can then run your tests and add coverage with the `--cov` option
- The `--cov` option can be:
  - a Python module: `python -m pytest --cov=add`
  - a Python package (use `.` for the current folder): `python -m pytest --cov=.`
- `coverage` will report about all the Python files used to run your program

Coverage reports

- Generate a nice looking HTML report: `python -m pytest --cov=. --cov-report=html`
- This will create / update contents in the `htmlcov` folder. You can open the `index.html` in your favorite browser.

Developing with a test-oriented mindset

Goals

- Make sure that tests are actually written
- Validate the requirements and design
- You become the 'user' of the code you are about to write
- You can work with stakeholders to resolve the anomalies/gaps

Issues

- Requires discipline
- "I am a developer, I want to code and not test"
- May appear useless at first sight

Unit testing: best practices

- Be reasonable
- No more test code than application code
- Code coverage of 80% is a good objective
- One minor change in the tests = one minor change in the application
- True for software development in general