The Art of Developer Testing

Aly Sivji
@CaiusSivjus

About Me

- Aly Sivji (Twitter: @CaiusSivjus)
- Mathematician / Software Engineer @ divvyDOSE
- Chicago Python Organizer
- Interests
  - Cycling
  - Data
  - Star Trek

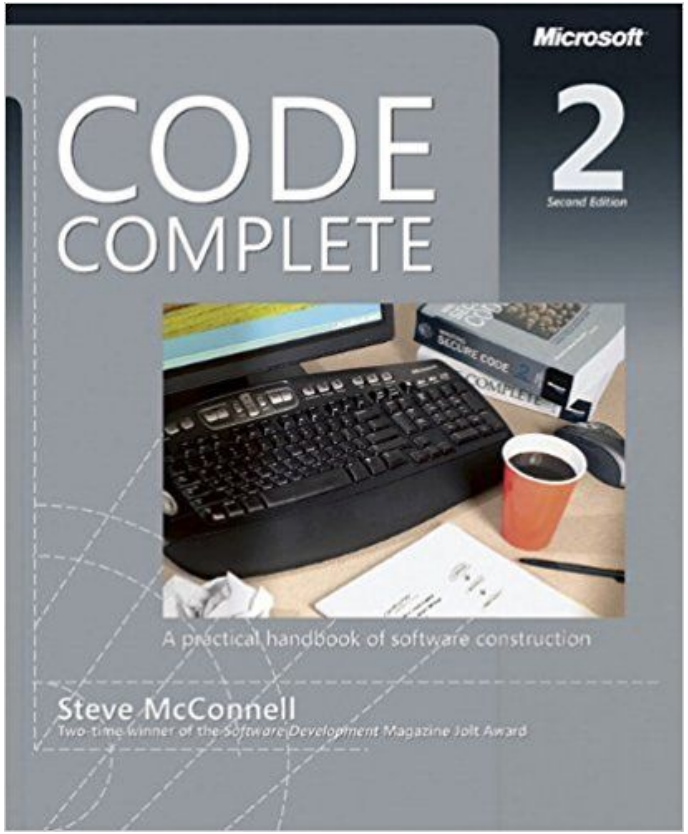
Agenda

- My Testing Journey
- WordCount-as-a-Service
- Testing Fundamentals
- pytest
- Testing Tools

I <3 Testing

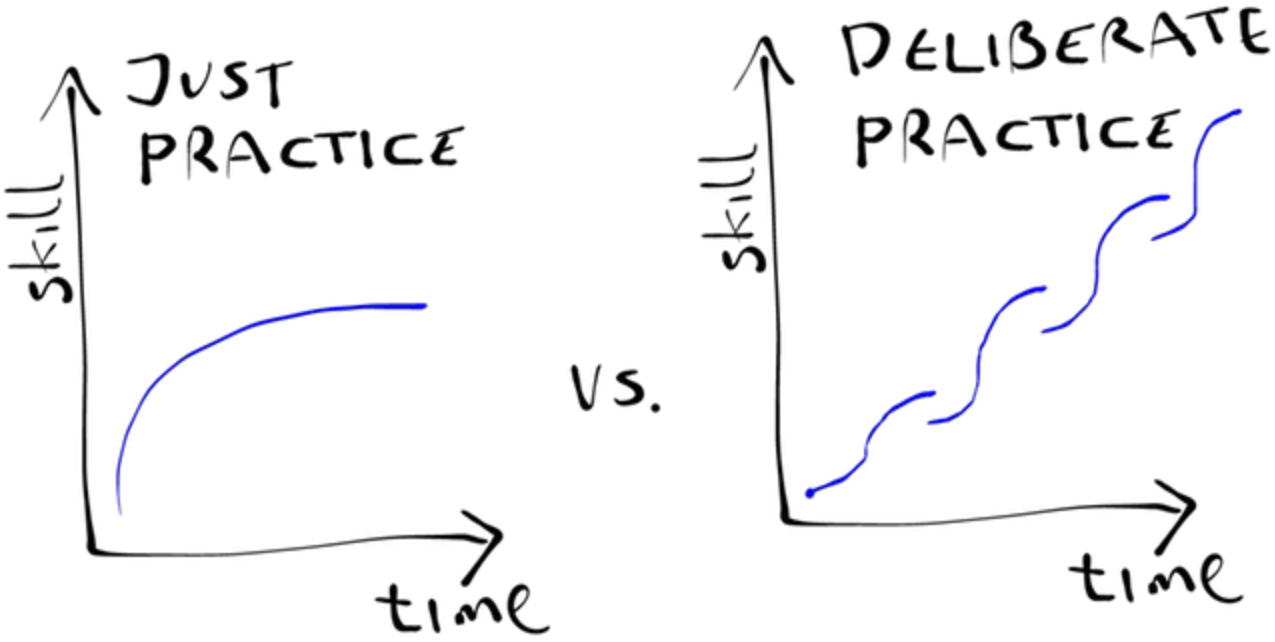
```
Database: jdbc:mysql://db:3306/sivdev (MySQL 5.7)
Successfully validated 1 migration (execution time 00:00.030s)
Creating Metadata table: `sivdev`.`schema_version`
Current version of schema `sivdev`: << Empty Schema >>
Migrating schema `sivdev` to version 0001 - Create movie table
Successfully applied 1 migration to schema `sivdev` (execution time 00:00.197s).
docker-compose exec api pytest --runslow
============================= test session starts ==============================
platform linux -- Python 3.6.4, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /app, inifile:
plugins: cov-2.5.1
collected 11 items

tests/resources/movies_test.py ...........                               [100%]

========================== 11 passed in 2.36 seconds ===========================
```


Where is my report? Your task delayed 2 days
I am a developer, not a Tester. I don't know how to test


Microsoft
CODE
COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award


COPYING AND PASTING
FROM STACKOVERFLOW
IS THIS SOFTWARE ENGINEERING?
imgflip.com

Source: Obie Fernandez



Source: PlusPlus


EFFICIENCY
QUALITY
SPEED
COST

Example Application

(WordCount-as-a-Service)

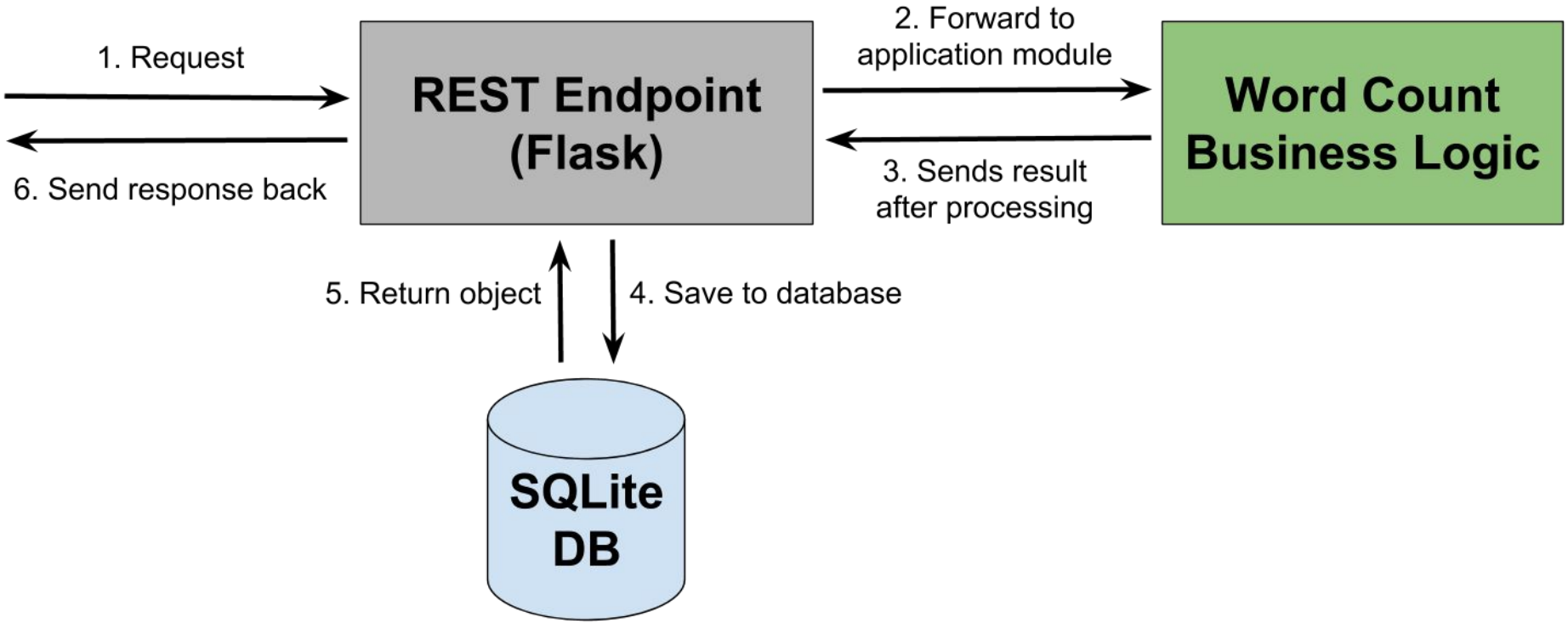
WordCount-as-a-Service

- REST endpoint that takes URLs and returns language statistics

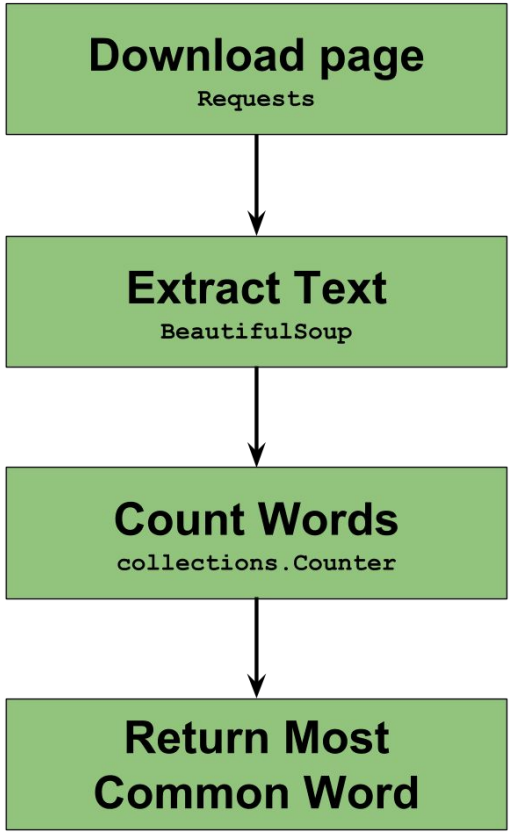
- `/top-word` endpoint as minimal viable product (MVP)
  - Returns JSON response with most common word and # occurrences

- Tests will allow us to refactor our code base without fear
  1. Build MVP
  2. Get funding
  3. ???
  4. ICO

Application Architecture



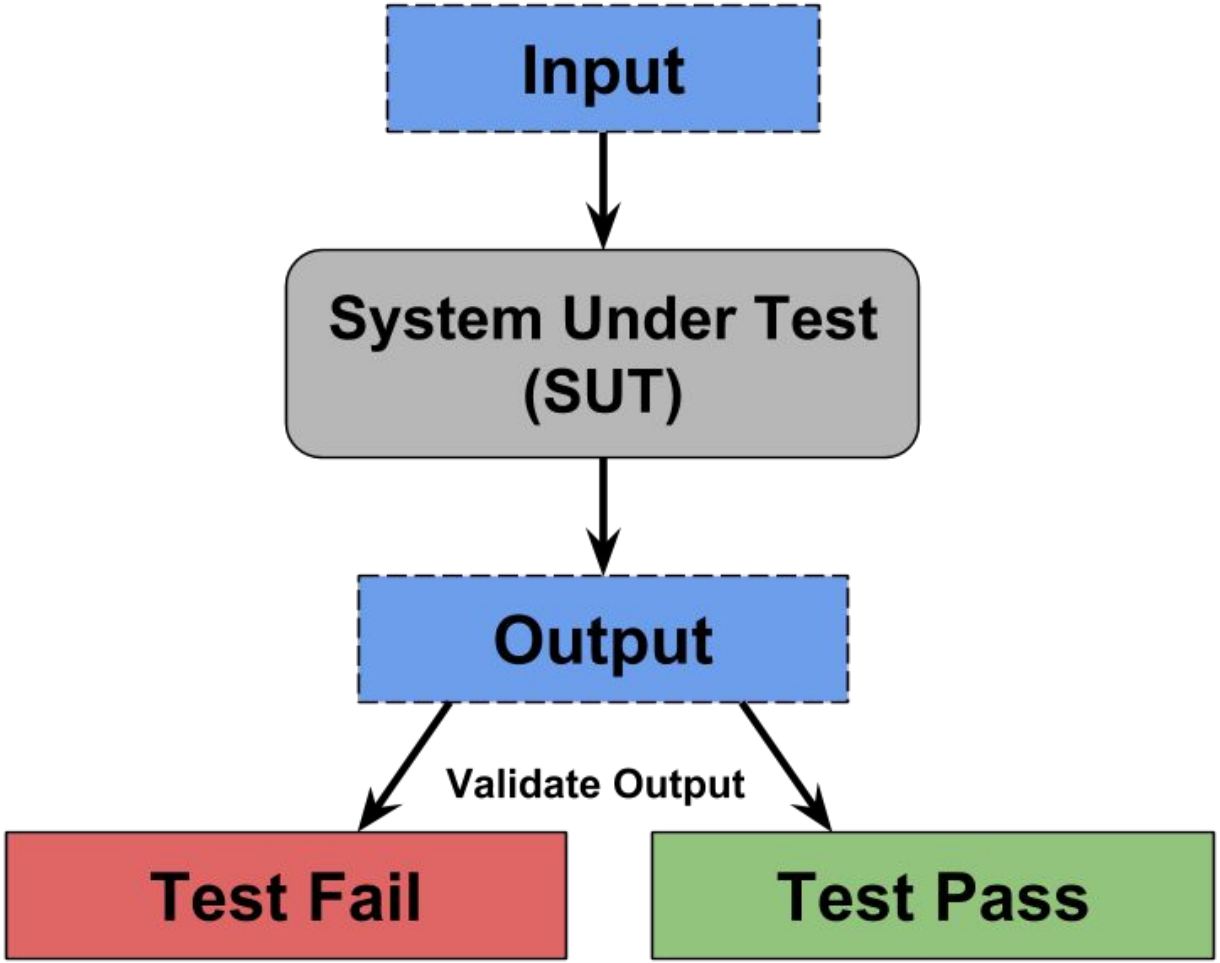
Business Logic



Application Code Walkthrough

- Walkthrough in VSCode

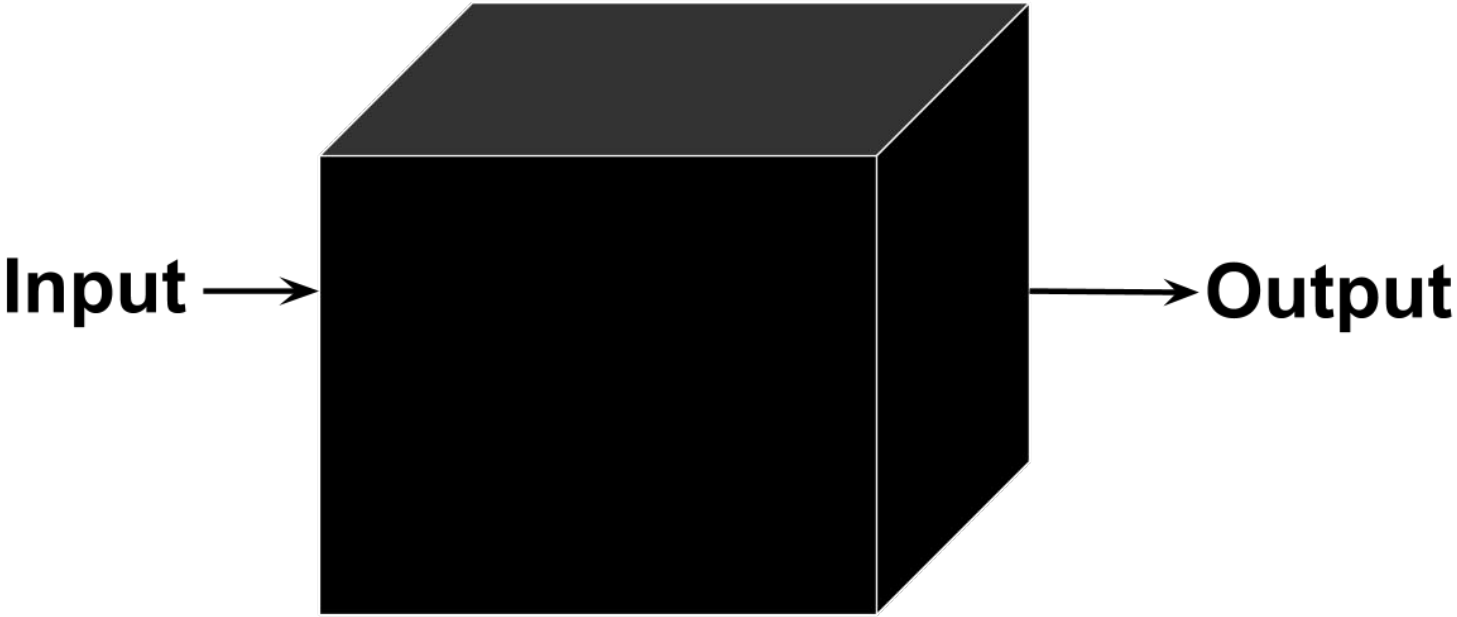
Testing Fundamentals


Input
System Under Test (SUT)
Output
Validate Output
Test Fail
Test Pass

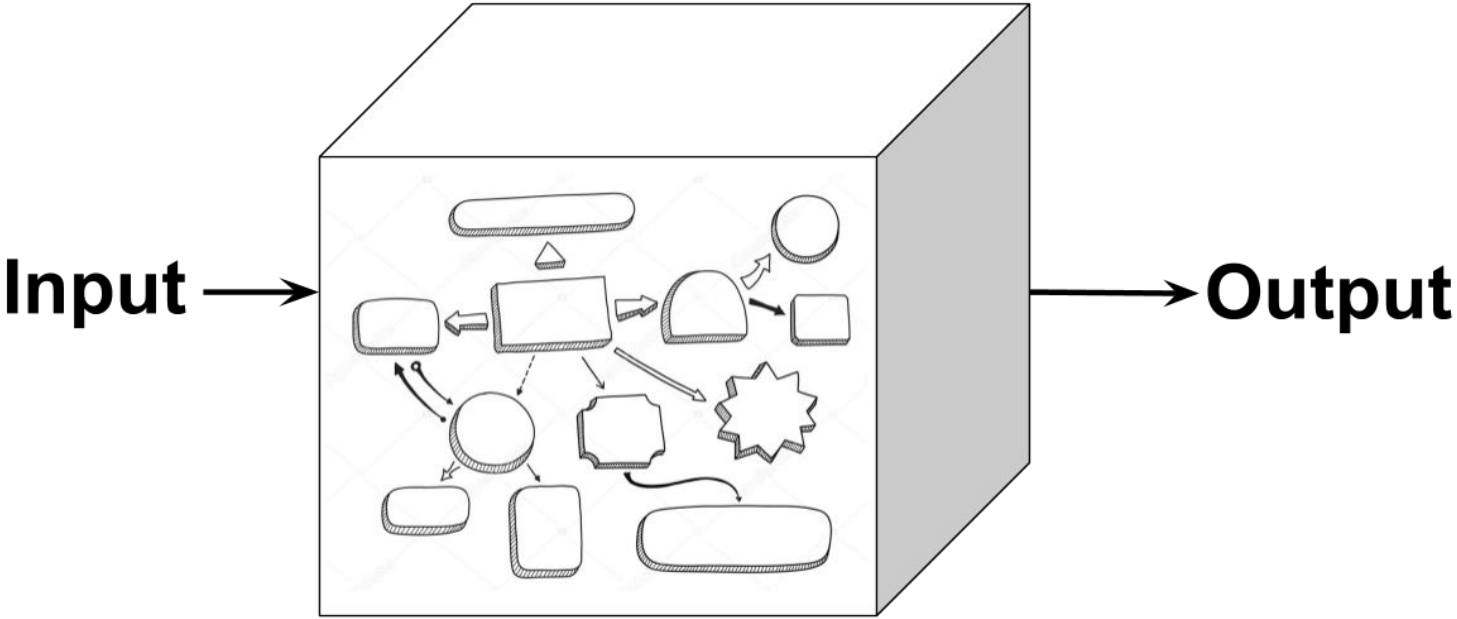
Types of Tests

- Acceptance
- Beta
- Functional
- End-to-End
- Integration
- Performance
- Regression
- Smoke
- Stress
- System
- Unit
- Usability

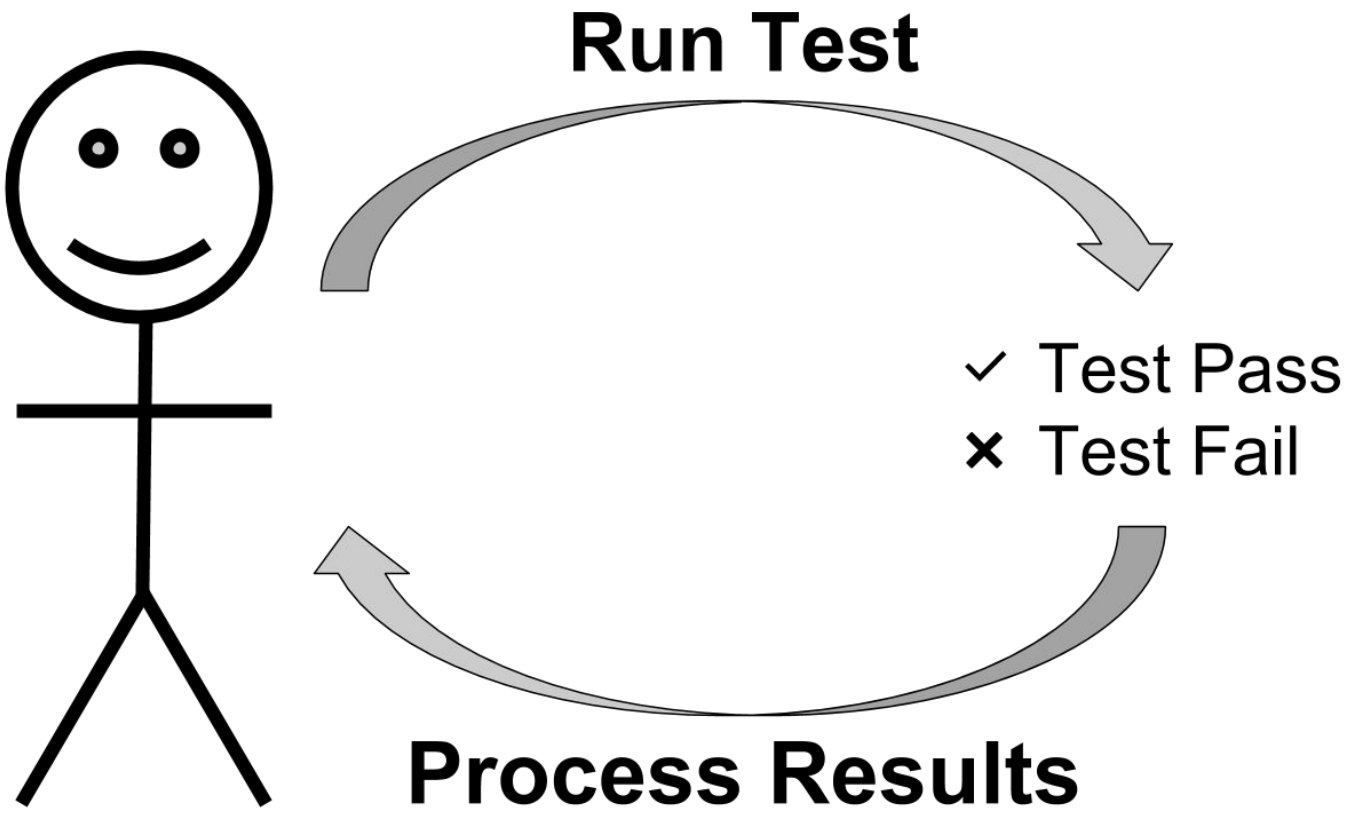
Black Box Testing



White Box Testing



Testing Feedback Loop



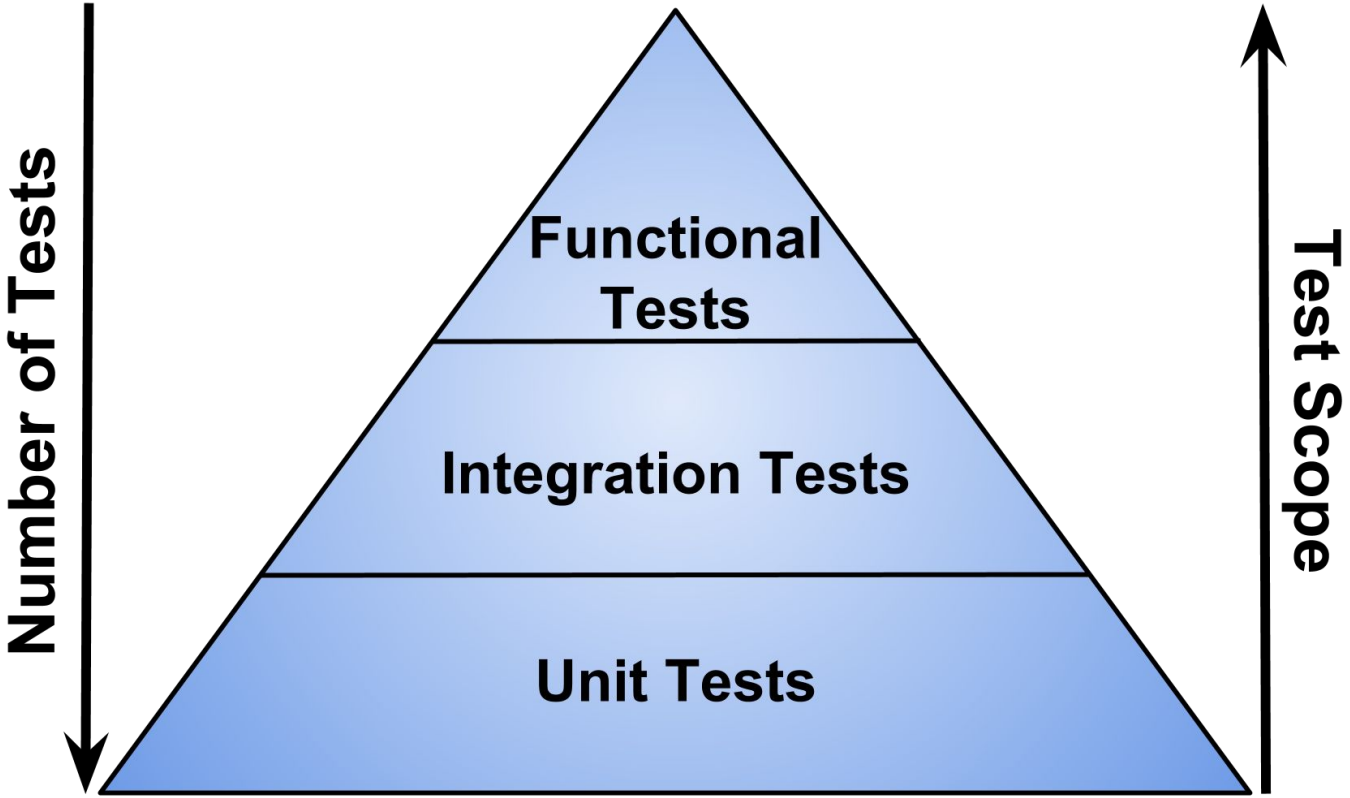
Developers are ~~lazy~~ efficient

Automated Testing

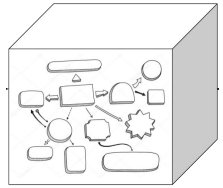
- Use software to control test execution and comparison of test output with expected result

- Benefits
  - Faster Feedback Loop
  - Reusability and Repeatability of Tests
  - Improve efficiency and use of resources
  - Integrate into Continuous Integration pipeline

Automated Testing Pyramid



Unit Tests



- Test individual units of code work as intended
- Unit is a testable part of an application
- Unit tests offer the most granular protection
  - Thoroughly test each function independently
  - Failing test pinpoints error

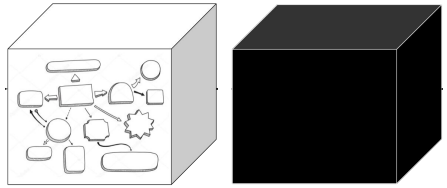
Unit Test -- Function To Test

```
def find_top_word(words)
    # Return most common word & occurrences
    word_counter = Counter(words)
    return word_counter.most_common(1)[0]
```

Unit Test -- Test Function

```
def test_find_top_word():
    words = ["foo", "bar", "bat", "baz",
             "foo", "baz", "foo"]

    result = find_top_word(words)

    assert result[0] == "foo"

    assert result[1] == 3
```

Integration Tests



- Combine multiple units and test as group
- Test integration boundaries with external services
  - Call out to database, filesystem, or external API
- Provide assurance components can work together
  - Failing test pinpoints integration point

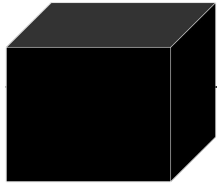
Integration Test -- Function To Test

```
def save_to_db():
    record = TopWord()
    record.url = url
    record.word = top_word[0]
    record.num_occurrences = top_word[1]

    db.session.add(record)
    db.session.commit()

    return record
```

Integration Test -- Test Function

```python
def test_save_to_db():
    url = "http://test_url.com"
    most_common_word_details = ("Python", 42)

    word = save_to_db(url, most_common_word_details)

    inserted_record = TopWord.query.get(word.id)
    assert inserted_record.url == "http://test_url.com"
    assert inserted_record.word == "Python"
    assert inserted_record.num_occurrences == 42
```

Functional Testing



- System is tested against functional requirements
- Conducted via UI or thru *subcutaneous* layer
- Verify program meets defined business requirements
  - Failing test can show where the process broke down
  - Fixing issue requires further investigation

Functional Test -- Test Function

```
def test_end_to_end():
    client = app.test_client()

    body = {"url": "https://meetup.com/indypy/"}
    response = client.post("/top-word", json=body)

    assert response.status_code == HTTPStatus.OK
```

Regression Testing

- Ensure bugs do not get reintroduced into system
- Good bug reports include code to reproduce errors
- Use bug report to create failing test case
  - Fix bug to fix failing test
  - Keep test to ensure program doesn’t regress
  - Document test case with related issue number

Benefits of Testing

- Validate the program works as expected
- Confirm changes do not break existing functionality
- Identify bugs earlier in the SDLC
  - Cheaper to find bugs early
- Force developers to write better code
  - Tests show developer has actually thought about problem

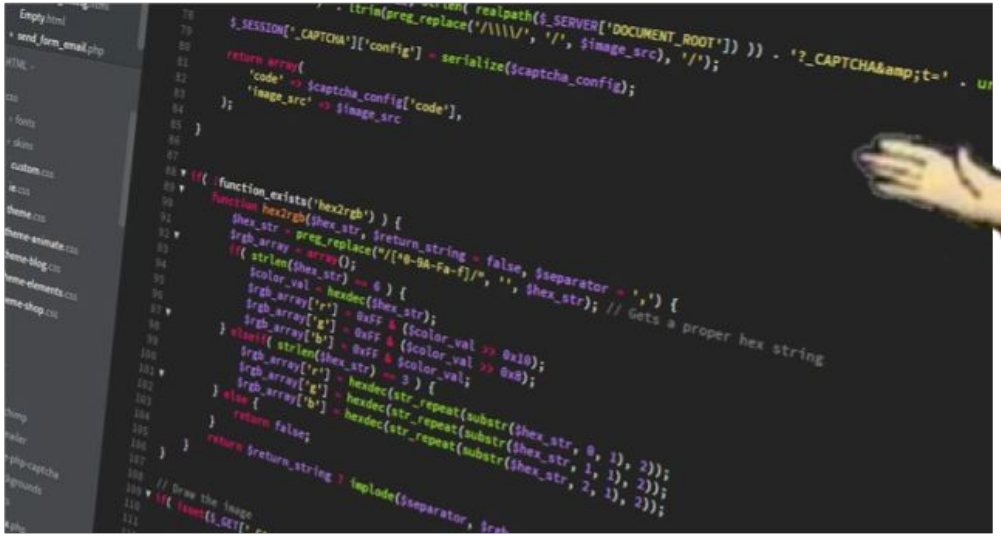
Test Metrics

- Test Ratio
- Test Speed
- Test Coverage

- When a measure becomes a target, it ceases to be a good measure (Goodhart's Law)

car salesman: *slaps roof of codebase*

this bad boy can fit 100% test coverage



Test Coverage

- Measures % of code base that is executed by test suite
  - Track lines of code that were exercised
  - Does not measure quality of tests
- Find untested code
  - Write tests around code
  - Delete code if not used
- Coverage tools calculate and reports test coverage
  - coverage.py

```
[1] MacSivPro:alysivji word-counter-as-a-service
 $   pytest --cov=app tests/
============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-3.6.3, py-1.5.4, pluggy-0.6.0
rootdir: /Users/alysivji/Documents/siv-dev/playground/word-counter-as-a-service,
 inifile:
plugins: xdist-1.22.2, forked-0.2, cov-2.5.1, hypothesis-3.65.0
collected 3 items

tests/functional_test.py .                                               [ 33%]
tests/integration_test.py .                                              [ 66%]
tests/unit_test.py .                                                     [100%]

---------- coverage: platform darwin, python 3.7.0-final-0 -----------
Name                   Stmts   Miss  Cover
----------------------------------------
app/__init__.py            0      0   100%
app/main.py               35      3    91%
app/word_counter.py       30      4    87%
----------------------------------------
TOTAL                     65      7    89%


========================== 3 passed in 2.29 seconds ==========================
```

pytest tests/ --cov=app --cov-report=html

Coverage for **app/main.py** : 91%

35 statements | 32 run | 3 missing | 0 excluded

```
1  import os
2  from typing import Tuple
3
4  from flask import Flask, jsonify, request
5  from flask_sqlalchemy import SQLAlchemy
6
7  from .word_counter import process_page_and_get_top_word
8
9  # Flask Configuration
10 app = Flask(__name__)
11 app.config["SQLALCHEMY_DATABASE_URI"] = os.getenv(
12     "DATABASE_URI", "sqlite:///../word_count.db"
13 )
14 app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
15 db = SQLAlchemy(app)
16
17
18 @app.shell_context_processor
19 def make_shell_context():
20     return {"app": app, "db": db}
21
```

Gaming Test Coverage

```python
# tests/coverage_example.py
def test_to_game_coverage():
    client = app.test_client()

    body = {"url": "http://test.com"}
    response = client.post("/top-word", json=body)

    # No Asserts
```

Gaming Test Coverage

```
tests/coverage_example.py .                                              [100%]

---------- coverage: platform darwin, python 3.7.0-final-0 -----------
Name                    Stmts   Miss  Cover
-----------------------------------------
app/__init__.py             0      0   100%
app/main.py                35      3    91%
app/word_counter.py        30      4    87%
-----------------------------------------
TOTAL                      65      7    89%
```

Additional Resources -- Testing Essentials

- Practical Test Pyramid
- Test and Code podcast: unit, integration, and system testing
- Martin Fowler Wiki: Test Coverage
- Podcast.__init__: Ned Batchelder on coverage.py

pytest

Background

Test Fixture

- Known state we want to run the test under to ensure repeat results
- Fixtures set up the test environment and return it to its original state

Test Case

- Individual unit of testing
- Checks that a specific input results in a specific output

Background

Test Suite

- Collection of test cases
- Aggregates tests that should be run together

Test Runner

- Orchestrates the execution and reporting of tests to user
- Fail tests include stack trace

Background

Invariant

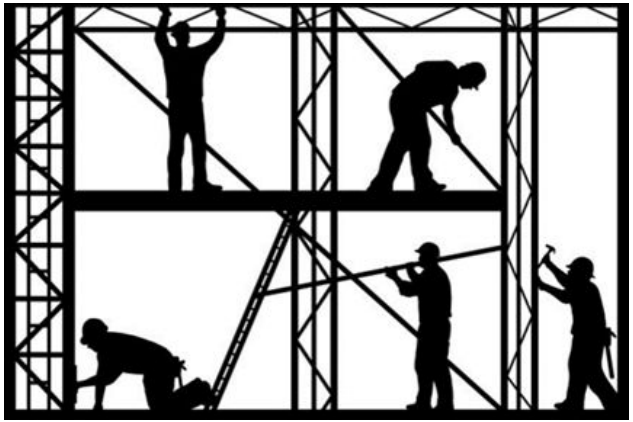
- Condition that is always `True`
- `assert` statement can be used to add invariants in Python (docs)
  - `AssertionError` if statement is `False`

```
In [1]: assert isinstance([1, 2, 3], list)

In [2]: assert isinstance([1, 2, 3], tuple)
---------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-2-7d6e159e119c> in <module>()
----> 1 assert isinstance([1, 2, 3], tuple)
```

Testing Framework

- Execution environment for automated tests

- Hook into / drive application under test
- Define how to express assertions
- Execute test cases
- Report results



Python Testing Frameworks

27.4. `unittest` —

Source code: Lib/unittest/__init__.py

- ✓ Part of Standard Library
- ✓ Familiar xUnit-style test pattern
- ✓ Extensive documentation
- ✗ Lots of Boilerplate code
- ✗ Feels like Java

pytest

- ✓ Doesn't feel like Java
- ✓ Tests are easy to read and write
- ✓ Extensive Documentation
- ✓ Plugin architecture
- ✓ Runs `unittest` test suites

pytest Features -- Test Assertions

- Test frameworks give detailed introspection about failed assertions
  - Difference between expected and actual output
- pytest uses **`assert`** statement syntax
  - No more **`assert*`** helper methods
- Additional Info
  - Write assertion about raising exceptions
  - Define custom assert comparisons

Source: pytest Docs

pytest Assert -- Failure Example

```
def test_failing_example():
    counter = range(10)
    my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 10]

    assert counter == my_list
```

pytest Assert -- Failure Example

```
========================================= FAILURES =========================================
_________________________________ test_failing_example _________________________________

    def test_failing_example():
        counter = range(10)
        my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 10]

>       assert counter == my_list
E       assert range(0, 10) == [0, 1, 2, 3, 4, 5, ...]
E         At index 9 diff: 9 != 10
E         Use -v to get the full diff

tests/test_playground.py:5: AssertionError
================================= 1 failed in 0.09 seconds =================================
```

pytest Features -- Fixture Model

- Fixtures are functions pytest runs before and after tests
  - Decorated with `@pytest.fixture`
- Test functions can use fixtures by naming them as input arguments
  - Searches current module then `conftest.py`
- Fixtures can be injected into other fixtures
  - Composition vs Inheritance
- Additional Info
  - Fixture Scope
  - Parameterized Fixtures

Source: pytest Docs

pytest Fixture -- Example

```
@pytest.fixture(scope="function")
def persisted_word():
    savepoint = db.session.begin_nested()
    db.session.begin_nested()

    word = TopWord(**{ param_dict })
    db.session.add(word)
    db.session.commit()

    yield word

    savepoint.rollback()
```

pytest Fixture -- Example

```
@pytest.fixture(scope="function")
def persisted_word():
    savepoint = db.session.begin_nested()
    db.session.begin_nested()

    word = TopWord(**{ param_dict })
    db.session.add(word)
    db.session.commit()

    yield word

    savepoint.rollback()
```

Set Up

Tear Down

pytest Fixture -- Inject into Test

```
def test_example_injection(persisted_word):

    # do tests here

    assert things_happened
```

fixture passed as parameter

pytest Pattern -- Fixture Factory

- Cannot pass arguments into fixtures
- Take advantage of Python language features to make fixture flexible

```
@pytest.fixture
def adder():
    def _wrapper(arg1, arg2):
        return arg1 + arg2
    return _wrapper
```

```
def test_function(adder):
    result = adder(1, 2)
    assert result == 3
```

- Additional Resources
  - Factories as Fixture pattern
  - Adding Function Arguments to pytest Fixtures

pytest Features -- Markers

- Add metadata to test functions via `@pytest.mark` decorator
  - `pytest -m 'selection'` runs selected tests
- [Implementation revamped in pytest 3.6]

```
@pytest.mark.api_test
def test_send_http_post():
    # business logic
    pass
```

```
[5] MacSivPro:alysivji word-counter-as-a-service
 $  pytest -m 'api_test'
================================ test session starts =============
platform darwin -- Python 3.7.0, pytest-3.6.3, py-1.5.4, pluggy-0.6
rootdir: /Users/alysivji/Documents/siv-dev/playground/word-counter-
:
plugins: xdist-1.22.2, forked-0.2, cov-2.5.1, hypothesis-3.65.0
collected 6 items / 5 deselected

tests/test_playground.py .

======================== 1 passed, 5 deselected in 0.04 seconds ===
```

Source: pytest Docs

pytest Features -- Markers

- Add metadata to test functions via `@pytest.mark` decorator
  - `pytest -m 'selection'` runs selected tests
- [Implementation revamped in pytest 3.6]

```
@pytest.mark.api_test
def test_send_http_post():
    # business logic
    pass
```

```
[5] MacSivPro:alysivji word-counter-as-a-service
 $  pytest -m 'api_test'
================================= test session starts =============
platform darwin -- Python 3.7.0, pytest-3.6.3, py-1.5.4, pluggy-0.6
rootdir: /Users/alysivji/Documents/siv-dev/playground/word-counter-
:
plugins: xdist-1.22.2, forked-0.2, cov-2.5.1, hypothesis-3.65.0
collected 6 items / 5 deselected

tests/test_playground.py .

======================== 1 passed, 5 deselected in 0.04 seconds ===
```

Source: pytest Docs

pytest Markers -- Builtin Markers

```
@pytest.mark.skip(reason="no way of currently testing this")
def test_the_unknown():
    ...
```

```
@pytest.mark.skipif(sys.version_info < (3,6),
                    reason="requires python3.6 or higher")
def test_function():
    ...
```

```
@pytest.mark.xfail
def test_function():
    ...
```

Source: pytest Docs

pytest Features -- Parameterized Fixtures

- Want to test function across many inputs
  - Test runner should treat each input as a different test case
- **`@pytest.mark.parametrized`** enables argument parameterization

```
import pytest
@pytest.mark.parametrize("test_input,expected", [
    ("3+5", 8),
    ("2+4", 6),
    ("6*9", 42),
])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

Source: [pytest Docs]

pytest Features -- Parameterized Fixtures

```
$ pytest
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 3 items

test_expectation.py ..F                                            [100%]

================================= FAILURES =================================
____________________________ test_eval[6*9-42] _____________________________

test_input = '6*9', expected = 42

    @pytest.mark.parametrize("test_input,expected", [
        ("3+5", 8),
        ("2+4", 6),
        ("6*9", 42),
    ])
    def test_eval(test_input, expected):
>       assert eval(test_input) == expected
E       AssertionError: assert 54 == 42
E        +  where 54 = eval('6*9')

test_expectation.py:8: AssertionError
==================== 1 failed, 2 passed in 0.12 seconds ====================
```

Source: [pytest Docs]

pytest Features -- Plugins

- Hook-based plugin architecture
- Local Plugins: `conftest.py` files within packages in test directory
  - Common fixtures shared to test package and subpackages
  - Fixtures in subpackages can override higher level fixtures
- Third Party Plugins: `pip install pytest-[name]`
  - Annotated list of popular plugins
  - Writing custom plugin
- Additional configuration via `pytest.ini`

Source: pytest Docs

pytest Features -- Test Runner

- Invoke runner using `pytest`
- Test Discovery
  - Files: `test_*.py` or `*_test.py`
  - Classes: `Test*`
  - Functions: `test_*`
- Run individual tests
  - `pytest test_module.py::TestClass::test_method`
  - `pytest -k "keyword_expression"`
- Test folder structure

Source: pytest Docs

WordCount-as-a-Service Revisited

- Jump to the terminal
- Show test runner options
  - pytest
  - pytest -m “debug_example” --pdb
  - Step thru PDB

`pytest` Command Line Options

```
-x                # stop after first failure

--maxfail=2       # stop after two failures

--pdb             # drop into pdb upon test failure

-v                # verbose mode

-q                # quiet mode
```

```
pytest --help
```

pytest Features -- Continuous Integration

- Add automated tests to continuous integration pipeline
- Running pytest can result in six different exit codes

  Exit code 0: All tests were collected and passed successfully
  Exit code 1: Tests were collected and run but some of the tests failed
  Exit code 2: Test execution was interrupted by the user
  Exit code 3: Internal error happened while executing tests
  Exit code 4: pytest command line usage error
  Exit code 5: No tests were collected

Source: pytest Docs

Additional Resources -- pytest

- Python Testing with pytest by Brian Okken
- pytest Documentation
- pytest API Reference
- awesome-pytest

Testing Tools

pytest

Tooling -- pytest Plugins

- builtin fixtures
- cov - measure code coverage
- freezegun - freeze time during test
- ipynb - test Jupyter notebook
- lazy-fixture - use fixture in parameterized tests
- webdriver - Selenium fixture
- xdist -- run tests in parallel

Tooling -- Test Doubles

- Python Standard Library **`unittest.mock`** module
  - Documentation
  - Examples
  - pytest-mock (wrapper around **`unittest.mock`**, available as **`mocker`** fixture)
- Mockito (port of popular Java mocking library)
- Pretend stubbing library
- response
  - Utility to mock Requests library

Postman

New | Import | Runner | My Workspace | Upgrade

Filter

History | Collections

- Aggregit — 0 requests
- appointment-manager — 8 requests
- Falcon-CRUD — 2 requests
- movie_recommendation_example — 3 requests
- Postman Echo — 21 requests
- thor — 23 requests
- thor_api_tests — 496 requests
- word-counter — 2 requests
  - GET health-check
  - POST website-word-counter

Heartbeat | Movies | health-check | health-check Copy | website-word-counte

No Environment

website-word-counter

Examples (0)

POST http://127.0.0.1:5000/top-word Params Send Save

Authorization | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code

form-data | x-www-form-urlencoded | raw | binary | JSON (application/json)

```
1 {
2     "url": "https://martinfowler.com/articles/practical-test-pyramid.html"
3 }
```

Body | Cookies | Headers (4) | Test Results

Status: 200 OK Time: 247 ms Size: 218 B

Pretty | Raw | Preview | JSON | Save Response

```
1 {
2     "id": 24,
3     "most_common_word": "the",
4     "num_occurrences": 454
5 }
```

Build | Browse

Selenium

More Testing Tools

- Data Science Testing
  - engarde
  - Great Expectations
- Database Testing
  - pgtap
- Property-based Testing
  - Hypothesis
- Mutation Testing
  - cosmic-ray

Aside: Writing Tests

- Test Structure
  - Given-When-Then
  - Arrange-Act-Assert
- Each test should be trying to test one specific thing
  - Failure should provide hints on how to fix
- Be pragmatic in what you test, how you test, and when you test
  - No reason for testing dogma
  - Writing Great Unit Tests: Best and Worst Practices
- Definition of unit can vary depending on the test you are doing
  - Unit of Behavior

Thank You

Github: alysivji/talks

Twitter: @CaiusSivjus

Blog: https://alysivji.github.io

Slides: http://bit.ly/art-of-developer-testing

Acknowledgements (Easter Egg)

- ChiPy

- AS, ES, CF, AS

Appendix

Slides below here do not fit into current form of presentation.

Writing Tests

- A failing test should let us know exactly what is wrong
  - If we assert something, let’s try to get more information vs less information
- What are we testing, what should be tested
- What to test, when to test, how to test
- Code complete, boundary values, etc
- Be pragmatic
  - Unit tests lead to better code design. I believe that. But code reviews can also do the same thing

¯_(ツ)_/¯

Slide on how to read pytest output

Size of a unit

- But the larger misconception, the one I find people really get hung up on when attempting to do TDD, is what they think a “unit” means. For most developers, when they see the term “unit” in “unit test,” they think of a unit of code, like a method or a block of statements, or even a single line of code. This is not what a “unit” means in this context. As I understand it, the term “unit” was adopted to emphasize a functionally independent unit of behavior. [1]

Assert one thing per test so you know what to fix when it goes wrong. This might be good for parameterized tests

Probably should be stubbing external dependencies with known values so we can test how our system handles input. But the database. In most application, that’s where all of our business value lies. So testing the database is important. Probably don’t want to mock that out.

Maybe a test that creates a new user and checks to make sure they are there.

Sometimes, you can write the same test at a higher level

The problem with test

We know that testing is important. We know we have to do it

It’s hard to determine what to test, how to test, and what makes a good test.

Code complete has good info (boundary value, tables, RCRCRC)

Thoughts on the Test Pyramid

Definition is subject

Honestly, what’s the real business value. Le’ts figure that out and test from there

Individual units aren’t the value add of our app. It’s how things work together and how we can take a process from start to finish as required. So maybe we need more functional and integration tests than the pyramid suggests

It’s good place to start