

Software Testing Workshop

Unit Testing, Test Coverage, and Mocking

Outline of the Workshop

- **Unit Testing and Test Coverage คืออะไร (5 minutes)**
- **ตัวอย่างการเขียน unit test (20 minutes)**
 - Pytest
 - Writing the first unit test
 - Fixtures in pytest
 - Mocking in pytest
 - Test Coverage with pytest-cov
- **งานเดี่ยวที่ นศ ต้องทำส่ง (50 minutes)**

What is Unit Testing

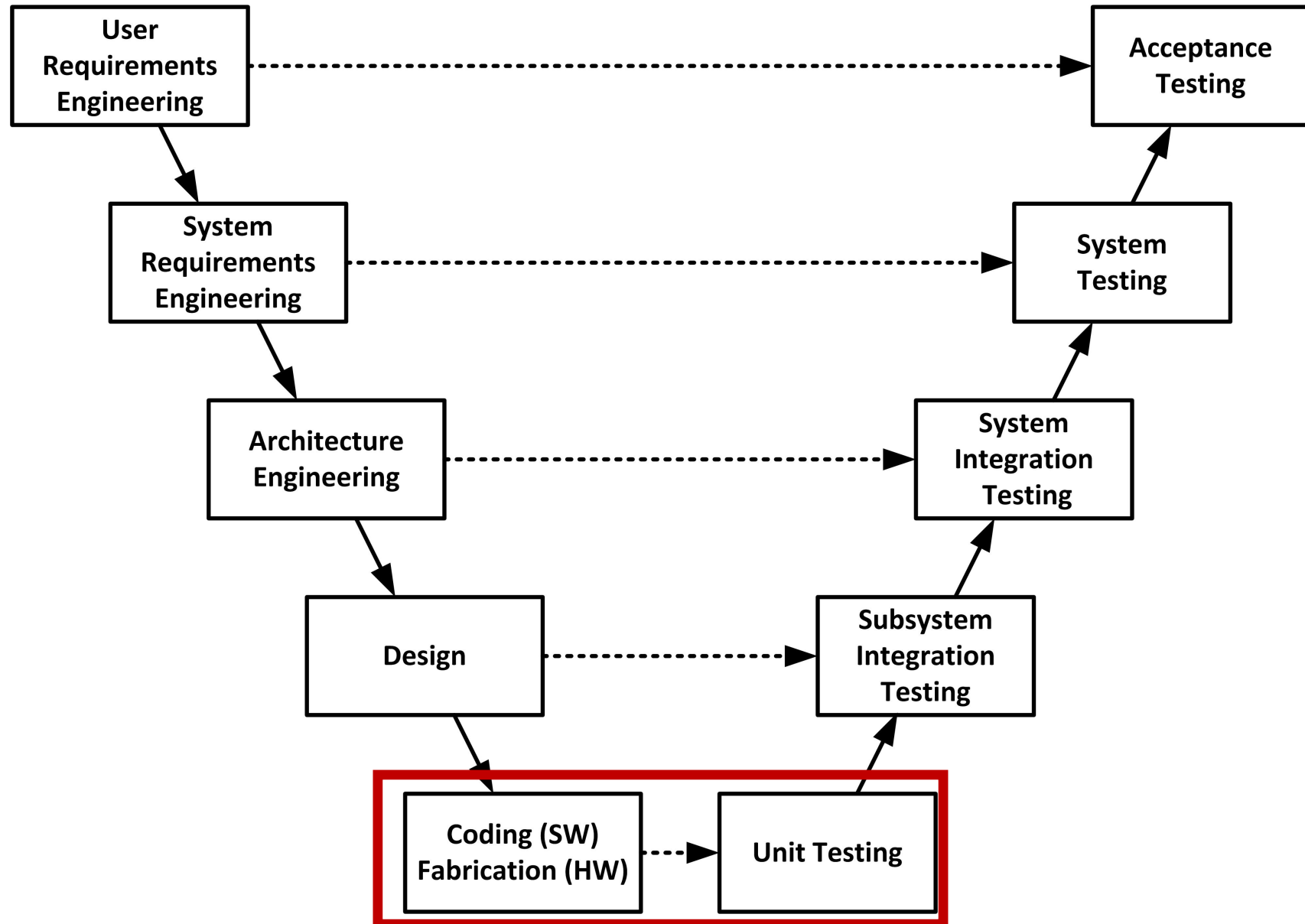
Does each unit work as specified?

- การทดสอบการทำงานของซอฟต์แวร์ในแต่ละหน่วยการทำงานที่เล็กที่สุดที่สามารถทดสอบได้ (มีการใช้งาน assert function และ สามารถเป็นได้ทั้ง White-box และ black-box)
- หน่วยในที่นี้อาจเป็น function, class, หรือ method หนึ่งๆ
- **ผู้ทดสอบคือ Development Team** ของระบบเพราะเข้าใจแต่ละส่วนมากที่สุด
- อาจต้องมีการสร้าง **Mock** มาช่วยจำลองด้านต่างๆ เช่น การเชื่อมต่อฐานข้อมูล (DB), การรับค่าข้อมูลเพื่อให้ unit ที่กำลังทดสอบสามารถทดสอบได้ (แต่ไม่ต้องไปต่อ service อื่นจริง ๆ – no dependency - **แบบว่าเอาตัวเองให้รอดก่อน**)

Why Unit Test?

- ช่วยค้นหาและลด **Fault** ที่จะก่อให้เกิดปัญหาหลง (ได้ feedback เร็วขึ้น)
- เพิ่มความมั่นใจให้กับผู้พัฒนาว่า หน่วยๆ ที่ทดสอบ ทำงานได้ตามที่คาดหวังไว้
- ลดเวลาและ **cost** ในแก้ไขปัญหาจาก **Fault** ในระยะยาว (ง่ายและถูกกว่า ถ้าแก้ไ้ดจาก Fault เน้นๆ ดีกว่ามาแก้ตอนท้าย ซึ่งก็จะไม่รู้ว่ามีผิดตรงไหน)
- เพิ่มความน่าเชื่อถือแก่ตัวโค้ดและระบบ
- เป็นคู่มือที่อ่านแล้วเข้าใจได้ทันทีว่า หน่วยๆ นี้ function การทำงานควรจะเป็นแบบไหน

When to Unit Test



Test Coverage

- **Test Coverage** บอกถึง จำนวน % ของโค้ดที่ได้รับการทดสอบ (เช่นจำนวน statement ที่ test ผ่าน, จำนวน branch ที่ test ผ่าน)
- **เพื่ออะไร?:** ประเด็นคือ การที่เราจะเจอ fault ได้ เราต้องผ่านมัน (ถ้ามันไม่ถูกใช้งาน เราก็ไม่รู้ได้เลยว่ามันเป็น fault หรือเปล่า)
- ช่วยให้เราเห็นว่า โค้ดของเราตรงส่วนไหนยังไม่ได้ถูกทดสอบ หรือ ถูกทดสอบไม่บ่อย ซึ่งอาจจะทำให้เกิด fault ได้ถ้าไม่ได้รับการทดสอบ

Today's Workshop



ทดสอบโค้ดที่เกี่ยวข้องกับระบบตะกร้าสินค้า



+ Pytest-Cov
(based on
Coverage)

+ Python's
unittest.mock

What You Will Learn

- การเขียน **unit test**
- การใช้ **assertion** เพื่อตรวจสอบ logic
- การใช้ **fixture** เพื่อ setup การทดสอบ
- การทำ **mock** เพื่อมาช่วยจำลองการทำงานของ dependency ที่หน่วยที่เราจะทดสอบต้องใช้งาน



ถึงแม้ว่าวันนี้เราจะทำ workshop ด้วย testing framework ที่ใช้เฉพาะกับ Python แต่ concept ที่จะได้ทำวันนี้สามารถนำไปใช้กับ testing framework อื่นและใช้ทดสอบระบบในภาษาอื่น ๆ ได้

Install Pytest and Pytest-cov

บน Terminal (mac) หรือ Command Prompt (Windows) พิมพ์

```
pip install pytest pytest-cov
```

Download Starter Code



getting_started

(ทำด้วยกัน)



your_turn

(สำหรับงาน workshop ทำเดี่ยว)

Download Link: <https://cmu.to/361TestingWorkshop1>

Running Pytest and Pytest-cov

Command to run pytest and pytest-cov:

```
pytest --cov --cov-branch --cov-report=html:myreport -v
```

Assertion

```
import pytest
```

Assertion

Assertion = statement ที่ใช้เพื่อทดสอบ condition ว่ามันเป็นจริงมั้ย
ถ้า assertion ได้ค่ากลับมาเป็น False นั้นหมายความว่า test ไม่ผ่าน

ตัวอย่าง


```
assert x == y  
  
assert item in some_list  
  
assert some_func() == number
```

Exception Handling in Pytest

- บางทีเราก็ต้องเช็ค ว่า โค้ดเรา raise exception ตามที่คาดหวังไว้มั้ย
- วิธีการทำใน pytest คือใช้ context manager “`pytest.raises`” ในการทดสอบ
- ตัวอย่าง: ต้อง `import pytest` ด้วยนะ

```
with pytest.raises(ValueError):  
    int("abc")
```

```
with pytest.raises(CustomException) as exc:  
    somefunction()  
assert str(exc.value) == "error message"
```

 เช็คเพิ่มเติมด้วยว่า error message ของ exception ตรงกับที่คาดหวังไว้หรือไม่

Fixture

Fixture

- **Fixture** ทำให้ได้หลายหน้าที่ ส่วนใหญ่เราจะใช้มันในการ setup สิ่งต่าง ๆ ก่อนเริ่มการทดสอบ และ teardown สิ่งต่าง ๆ หลังการทดสอบ
 - เช่น Login ผู้ใช้ไว้ก่อน, สร้าง Object ของคลาสนี้ขึ้นมาก่อน เป็นต้น
- **Fixture** ช่วยให้เราสามารถใช้ function ที่เป็น Fixture นั้นในหลายๆ test ได้ (reusability)
- **Fixture** ยังช่วยในการรันการทดสอบโดยใช้หลายค่า input ได้อีกด้วย


```
@pytest.fixture
def createInstance():
    #setup phase before yield
    p = someClass()
    yield p
    #teardown goes after "yield"
```

```
def test_something(createInstance):
    instance = createInstance
    ...
```

```
def test_another_thing(createInstance):
    instance = createInstance
    ...
```

ก่อน yield คือ setup
phase หลังจาก test
อื่นใช้ มัน โค้ดล่าง
yield จะทำงานเป็น
teardown phase

สามารถเอา Fixture
ไปใช้ได้หลาย test
เพื่อให้มัน setup
และ teardown ให้

```
@pytest.fixture(params=[(1,2,3), (4,5,9), (10,20,30)])
```

```
def set_of_values(request):
    return request.param
```

```
def test_addition(set_of_values):
    a, b, expected = set_of_values
    result = add(a, b)
    assert result == expected
```

request object ใช้
สำหรับ access ค่าที่
ใส่ไว้ใน fixture

Test อื่นสามารถเอา
Fixture นี้ไปใช้ได้เพื่อ

แทนค่าต่างๆ

Fixture to Capture stdout stderr Output

```
def some_function():  
    ...  
    print("some ok text")  
    ...  
    sys.stderr.write("some err text")
```

ใช้ **Fixture** ที่ชื่อว่า **capsys** ในการเช็คค่า **stdout** และ **stderr**

```
def test_some_function(capsys):  
    module.some_function()  
    captured = capsys.readouterr()  
    assert "some ok text" in captured.out  
    assert "some err text" in captured.err
```

Mock

```
from unittest.mock import Mock, patch
```

Mock with Python's Unittest Module

- **Mocking an Object**


- เปลี่ยน object ที่โค้ดเราต้องใช้ให้เป็น Mock object เพื่อที่จะเราจะสามารถกำหนดค่า return value ของมันได้ (โดยที่เราไม่ต้องไปพัฒนาตัว object นั้นจริงๆ หรือทดสอบ object นั้นๆ)

เช่น โค้ดเราต้องใช้ database object ที่ส่งมาเป็น argument

```
def get_username_by_id(database, user_id):  
    return database.fetch(user_id)
```

เราสามารถทดสอบโค้ดเราโดยไม่ต้องพึ่ง database object จริงๆ แต่ใช้ mock object แทน

```
def test_get_username_by_id():  
    db_object = Mock()  
    db_object.fetch.return_value = 'Andrew'  
    assert get_username_by_id(db_object, 3) == 'Andrew'  
    db_object.fetch.assert_called_once_with(3)
```

 ตรวจสอบว่า fetch ถูกเรียกใช้งานหนึ่งครั้งด้วย argument ที่มีค่าเท่ากับ 3

Mock with Python's Unittest Module

- Mocking a class and an instance with **@patch**

- ในกรณีที่โค้ดเรามีการเรียกใช้ Class อื่น เราก็สามารถสร้าง Mock class นั้นๆขึ้นมาได้ด้วย patch

ใช้ **patch** ในการเปลี่ยน

DatabaseClient() ใน file **demo**

ให้เป็นชื่อ **mock_db** (ตั้งชื่ออื่นก็ได้)

File ชื่อว่า **demo.py**

```
from db_client import DatabaseClient
```

```
def get_user_data(user_id):  
    db = DatabaseClient()  
    db.connect()  
    data = db.fetch_data(user_id)  
    db.close()  
    return data
```

```
@patch('demo.DatabaseClient')  
def test_get_user_data(mock_db):  
    instance = mock_db.return_value #ต้องสร้าง instance ของ mock_db ก่อน  
  
    instance.fetch_data.return_value = {"id": 1, "name": "Andrew"}  
  
    data = get_user_data(1)  
    assert data == {"id": 1, "name": "Andrew"}  
    ...
```

Mock with Python's Unittest Module

- Mocking with **side effect**

- บางที่เราอยากให้ return value เปลี่ยนในแต่ละ method call หรือเปลี่ยนโดยขึ้นอยู่กับค่า argument ส่งเข้ามาใน method เราต้องใช้ **side effect**

เปลี่ยนจาก return_value เป็น side_effect

```
mock_obj.method.side_effect = [2, 4, 6]
```

method นี้ของ mock_obj จะคืนค่า 2 เมื่อถูกเรียกครั้งแรก, 4 เมื่อถูกเรียกครั้งที่สอง, และ 6 เมื่อถูกเรียกครั้งที่ 3 (หากถูกเรียกเกิน 3 ครั้งจะ error)

เราสามารถคืนค่าที่แตกต่างขึ้นอยู่กับ argument ที่ผ่านเข้ามาใน method

```
def mock_function(arg):  
    if arg == 1:  
        return 2  
    elif arg == 2:  
        return 4  
    else:  
        return 6
```

```
mock_obj.method.side_effect = mock_function
```

Method นี้จะคืนค่าตามที่เรากำหนดโดยขึ้นอยู่กับแต่ละ argument ที่ผ่านเข้ามาใน mock_function

User Story #1

As a user, I can add items to the cart.



Acceptance Criteria:

- ผู้ใช้ไม่สามารถเพิ่มสินค้ามากกว่า 5 ชนิดที่แตกต่างกันลงในตะกร้าได้ หากจะเพิ่มเกินต้องมีการแจ้งเตือนผู้ใช่ว่าไม่สามารถทำได้
- ผู้ใช้ไม่สามารถที่จะเพิ่มสินค้าได้เกินชนิดละ 10 ชิ้น หากเกินต้องมีการแจ้งเตือนผู้ใช่ว่าไม่สามารถทำได้
- ผู้ใช้ต้องทำการยืนยันตัวตนกับระบบด้วย username และ password ที่ถูกต้อง ก่อนถึงจะสามารถนำสินค้าลงตะกร้าได้ หากผู้ใช้พยายามจะนำสินค้าลงตะกร้าโดยไม่ได้ยืนยันตัวตน ระบบต้องแจ้งเตือนผู้ใช้

User Story #2

As a user, I can see the total price of items in the cart.



Acceptance Criteria:

- ผู้ใช้สามารถดูราคารวมของสินค้าทั้งหมดในตะกร้า ที่คิดจากราคารวมของราคาสินค้าคูณกับจำนวนสินค้าแต่ละชนิดในตะกร้าได้
- หากผู้ใช้ไม่มีสินค้าในตะกร้าให้คืนค่า 0 กลับมา
- ราคาของสินค้าแต่ละชนิดจะได้มาจาก Database ที่เก็บข้อมูลราคาสินค้าไว้ และแต่ละชนิดราคาจะไม่เหมือนกัน
- ผู้ใช้ต้องทำการยืนยันตัวตนกับระบบด้วย username และ password ที่ถูกต้อง ก่อนถึงจะสามารถดูราคารวมของสินค้าในตะกร้าได้ หากผู้ใช้จะดูราคารวมของสินค้าโดยไม่ได้นับยืนยันตัวตน ระบบต้องแจ้งเตือนผู้ใช้