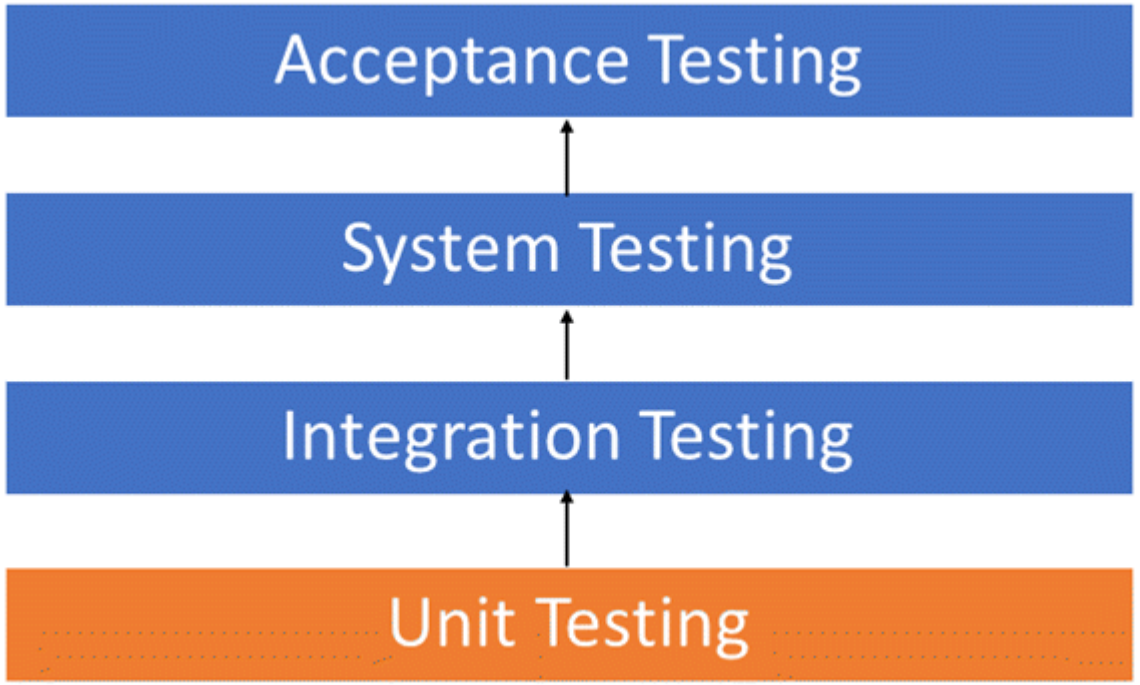
Unit Testing

Hönnun og smíði hugbúnaðar

Haust 2022

Hvað er Unit Testing

- Unit er lítill partur af kóða
  - Föll
  - Modules eða klasar
  - Lítill hópur af skyldum klösum
  - Etc.
- Uni Test skilgreining
  - Unit Test eru prófanir fyrir lítinn isolated part af kóða(i.e unit) til að athuga hvort að sá kóðabútur mætir þeim hönnunar kröfum sem er ætlast til af honum
  - þ.e.a.s Unit Test athuga/test-a hvort kóðabúturinn keyrir rétt án villna
  - Skilgreinir samning sem ákveðinn kóðabútur verður að fara eftir
- Gert af forritaranum sem bjó til unit-ið
  - Unit testing er talið vera partur af programming fasanum (sjá TDD)
  - Í öðrum test-um er það oft einhver annar en forritarinn sem hannar þau test
- Automated Unit Tests
  - Unit Tests eru yfirleitt automated
  - Hannað af mönnum (þ.e.a.s við skrifum unit test-in)
  - Keyrir án mannlegra afskipta
  - Skilar frá sér annað hvort Pass eða Fail (og mögulega með einhverjum error skilaboðum þegar test-ið fail-ar)
  - Gerir kleift að test-a allt kerfið mjög fljótt
  - Gerir hluti eins og Continuous Integration/Deployment mögulegt
- Unit test eru einföld
  - Nota yfirleitt ekki filesystem-ið
  - Nota ekki gagnagrunn
  - Nota ekki netið
  - Keyra eingöngu í minni
  - Eiga að keyra mjög hratt
  - Hvert Unit Test ætti að test-a eina virkni
    - gætir t.d. haft mörg unit test fyrir eitt fall
    - Unit Test ættu ekki að hafa dependency á önnur unit (hægt að gera með Mocks, Stubs etc.)
  - Unit test ættu að vera stutt


Acceptance Testing
System Testing
Integration Testing
Unit Testing

Unit Tests kostir

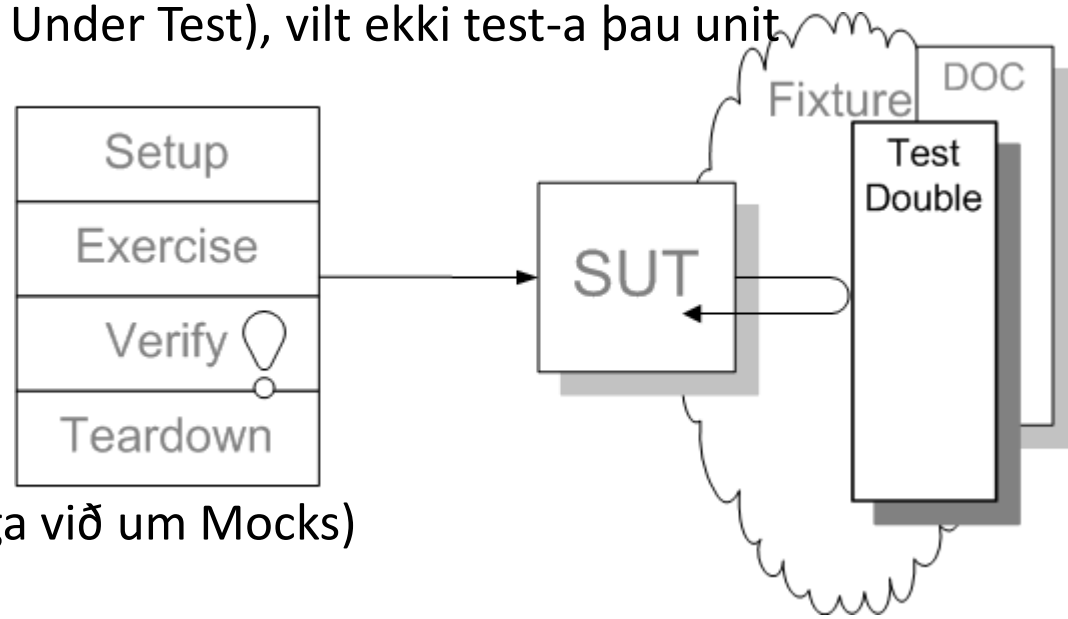
- Finna vandamál snemma
  - Villur í kóðanum
  - Vandamál við lýsinguna á unit-inu sjálfu
  - Fail fast
    - Vilt ekki finna í Acceptance Testing, eða í production
- Eru ódýr
  - Ódýrasta tegund af testum
  - Ódýrt að skrifa
  - Keyra hratt
  - **Að testa er dýrt en að testa ekki er dýrara**
- Neyða mann til að skrifa góðan kóða
  - Illa skrifaður og tightly coupled kóði er martröð að test-a
  - Testing neyðir mann að skrifa loosely coupled code úr litlum einangruðum einingum sem er auðvelt að test-a
- Gefa öryggi við breytingar
- Virka sem documentation
  - Lýsa hvað unit-ið á að gera
- Hjálpa forritum að skilja hvað þeir eru að smíða
  - Hjálpa að finna göt í skilning á vandamálinu
  - Láta okkur hugsa um smáatriðin

Unit Tests ókostir og takmarkanir

- Munt aldrei geta ekki gripið allar villur
  - Getur ekki test-að alla möguleika
  - Kóði keyrist ekki í raun umhverfi
  - Test cases geta verið önnur en í raun umhverfinu
- Prófa bara kóðann í unit-inu
  - Grípa ekki integration villur
  - test-a ekki non-functional requirements
  - Aðrar tegundir af prófunum fyrir það
    - Acceptance Testing
    - Integration Testing
    - Performance Testing
    - etc.
- Geta líka haft villur
  - Unit Test eru kóði og alveg eins og allur annar kóði geta verið villur í unit test-unum
- Sumt er erfitt að prófa
  - Threading og concurrency
  - Vista í Gagnagrunn
  - Etc.
- Erfitt að gera góð og raunsæ test
  - Þarft að setja upp relevant og realistic aðstæður áður en test-ið er keyrt
- Tekur orku og tíma að test-a allan kóðann
  - Krefjast viðhalds
  - Nýr kóði -> Ný test

Test Doubles

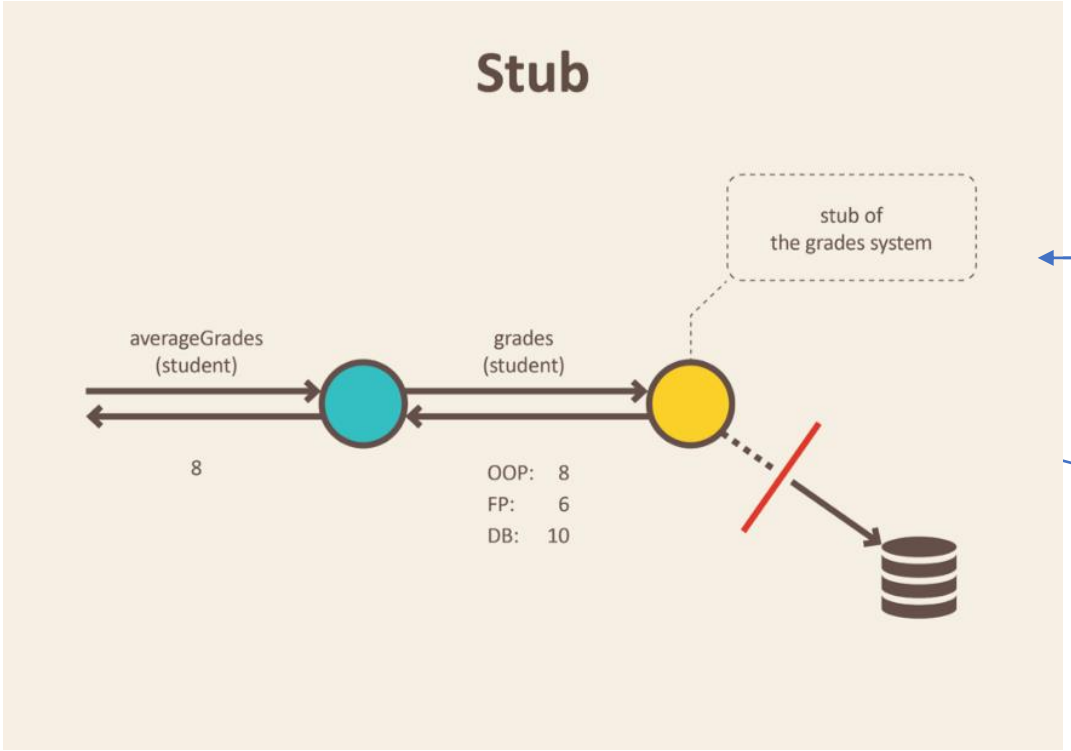
- Test double er stand-in fyrir alvöru hlut
  - Test double er generic term fyrir stand-in af alvöru hlut í test-um
  - Hluturinn sem er verið að test-a(SUT) ætti ekki að vita að hann er að tala við Test Double
- Af hverju Test Double?
  - Vilt bara test-a virknina í einu unit-i þ.e.a.s SUT-ið (System Under Test), vilt ekki test-a þau unit sem SUT-ið depend-ar á
  - Oft erfitt að test-a alvöru hluti
    - Þau depend-a á alvöru resources (API, gagnagrunn…)
    - Resource intensive og hægir
    - Ekki tiltækur (t.d. óútfærður)
    - Óæskilegar aukaverkanir
    - Þurfum meiri stjórn eða meira visibility



- Test Double er stillanleg
  - Allt sem Test Double gerir er stjórnað af test-inu (á aðallega við um Mocks)
- Tegundir
  - Stub, Spy, Mock, Fake, Dummy
- Gallar
  - Testum ekki alvöru integration

Test Stub

- Stub útfærir sama interface
- Stub hefur einfaldari virkni en alvöru hluturinn



Grading Stub hefur einfaldari útfærslu af Grading kerfinu með eitthverjum harðkóðuðum output-um til að test-a

Skilagildi úr föllum eru harðkóðuð inn

Stub dæmi

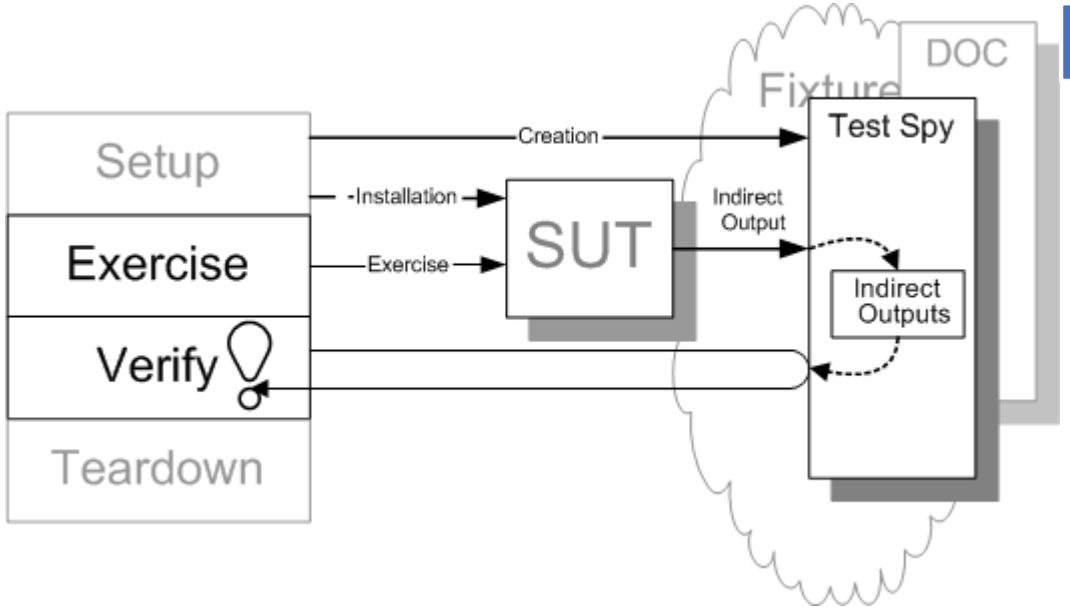
```python
class IOrderApiGateway(ABC):
    @abstractmethod
    def get_order(self, order_id: int) -> Optional[Order]:
        pass

    @abstractmethod
    def post_order(self, order: Order) -> int:
        pass
```

```python
class OrderApiGateway(IOrderApiGateway):
    def __init__(self, base_url: str):
        self.__base_url = base_url

    def get_order(self, order_id: int) -> Optional[Order]:
        response = requests.get(f'{self.__base_url}/orders/{order_id}')
        if response.status_code == HTTPStatus.NOT_FOUND:
            return None
        elif response.status_code == HTTPStatus.OK:
            return Order(**response.json())

        raise Exception('api exception', response)

    def post_order(self, order: Order) -> int:
        response = requests.post(f'{self.__base_url}/orders', order.json())
        return response.json()['id']
```

```python
class OrderApiGatewayStub(IOrderApiGateway):
    def get_order(self, order_id: int) -> Optional[Order]:
        return Order(id= self.__next_id(),
                     name='order name',
                     price=100,
                     buyer_name='john',
                     merchant_name='johnsson')

    def post_order(self, order: Order) -> int:
        pass

    def __next_id(self):
        return 1
```

Test Spy

- Spies eru Stubs með *recording* eiginleika fyrir test-in
- Gerir mögulegt að sannreyna óbeinar afleiðingar
- T.d. EmailServiceSpy er með einfaldaða virkni(en sama interface) en hefur þá auka virkni að halda utan um þann fjölda af email-um sem er sent út sem er hægt að verify-a í testum



DOC = depended-on component

Spy dæmi

```python
class IOrderApiGateway(ABC):
    @abstractmethod
    def get_order(self, order_id: int) -> Optional[Order]:
        pass

    @abstractmethod
    def post_order(self, order: Order) -> int:
        pass
```

```python
class OrderApiGateway(IOrderApiGateway):
    def __init__(self, base_url: str):
        self.__base_url = base_url

    def get_order(self, order_id: int) -> Optional[Order]:
        response = requests.get(f'{self.__base_url}/orders/{order_id}')
        if response.status_code == HTTPStatus.NOT_FOUND:
            return None
        elif response.status_code == HTTPStatus.OK:
            return Order(**response.json())

        raise Exception('api exception', response)

    def post_order(self, order: Order) -> int:
        response = requests.post(f'{self.__base_url}/orders', order.json())
        return response.json()['id']
```

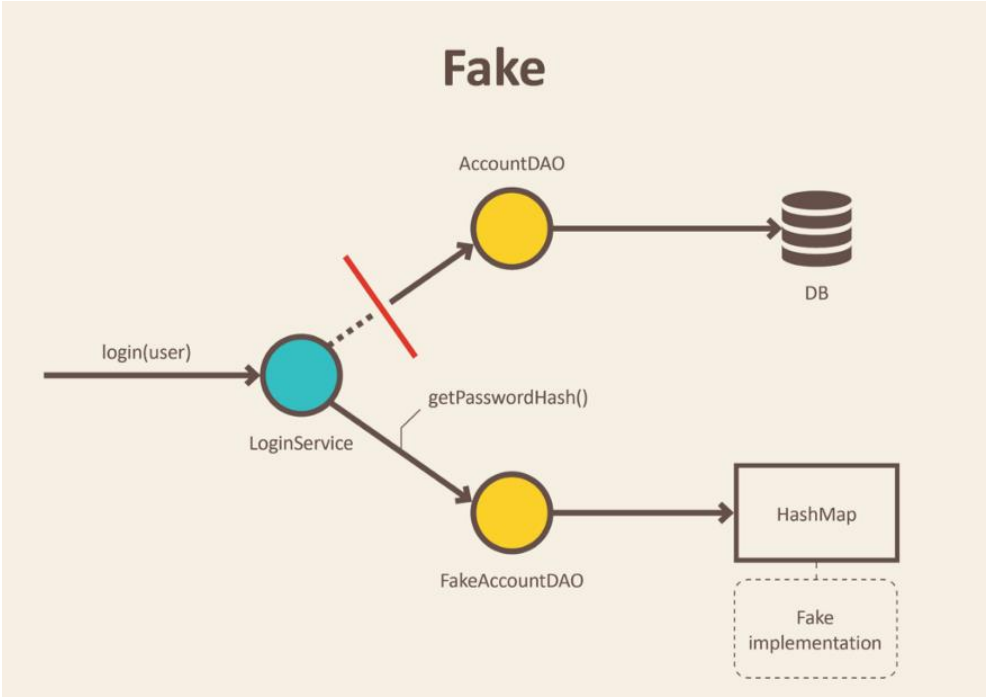
```python
class OrderApiGatewaySpy(IOrderApiGateway):
    def __init__(self):
        self.posted_count = 0

    def get_order(self, order_id: int) -> Optional[Order]:
        return Order(id= self.__next_id(),
                     name='order name',
                     price=100,
                     buyer_name='john',
                     merchant_name='johnsson')

    def post_order(self, order: Order) -> int:
        self.posted_count += 1

    def __next_id(self):
        return 1
```

Test Fake

- Fake hlutir útfæra sama interface og alvöru hluturinn

- Fake hafa alvöru virkni en taka shortcuts
  - Ekki production hæft
  - T.d. in-memory gagnagrunnur

- Getur verið gagnlegt fyrir development umhverfi
  - Vilt ekki nota raun gögn
  - Vilt ekki tala við raun API
  - Þegar alvöru hluturinn er resource intensive
  - Þegar þú vilt ekki vista gögn á dev umhverfi
  - Etc.



Fake dæmi

```python
class IOrderApiGateway(ABC):
    @abstractmethod
    def get_order(self, order_id: int) -> Optional[Order]:
        pass

    @abstractmethod
    def post_order(self, order: Order) -> int:
        pass
```

```python
class OrderApiGateway(IOrderApiGateway):
    def __init__(self, base_url: str):
        self.__base_url = base_url

    def get_order(self, order_id: int) -> Optional[Order]:
        response = requests.get(f'{self.__base_url}/orders/{order_id}')
        if response.status_code == HTTPStatus.NOT_FOUND:
            return None
        elif response.status_code == HTTPStatus.OK:
            return Order(**response.json())

        raise Exception('api exception', response)

    def post_order(self, order: Order) -> int:
        response = requests.post(f'{self.__base_url}/orders', order.json())
        return response.json()['id']
```

```python
class OrderApiGatewayFake(IOrderApiGateway):
    def __init__(self):
        self.__orders = []

    def get_order(self, order_id: int) -> Optional[Order]:
        try:
            return next(order for order in self.__orders if order.id == order_id)
        except StopIteration:
            return None

    def post_order(self, order: Order) -> int:
        order_copy = Order(**order.dict())
        order_copy.id = self.__next_id()
        self.__orders.append(order_copy)
        return order_copy.id

    def __next_id(self):
        if len(self.__orders) == 0:
            return 0
        else:
            return max([order.id for order in self.__orders] ) + 1
```

Test Dummy

- Heimskasti test double
- Passed around en aldrei notaður
- Aðallega ætlaðir til að fylla einhvern parameter list-a

Dummy dæmi

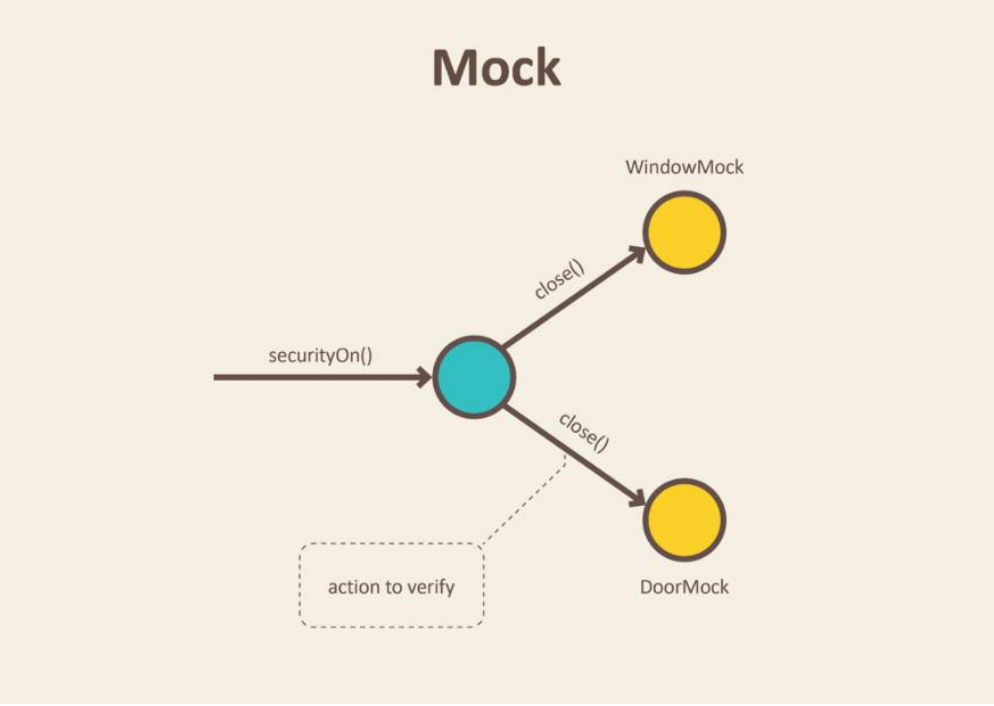
```python
class IOrderApiGateway(ABC):
    @abstractmethod
    def get_order(self, order_id: int) -> Optional[Order]:
        pass

    @abstractmethod
    def post_order(self, order: Order) -> int:
        pass
```

```python
class OrderApiGateway(IOrderApiGateway):
    def __init__(self, base_url: str):
        self.__base_url = base_url

    def get_order(self, order_id: int) -> Optional[Order]:
        response = requests.get(f'{self.__base_url}/orders/{order_id}')
        if response.status_code == HTTPStatus.NOT_FOUND:
            return None
        elif response.status_code == HTTPStatus.OK:
            return Order(**response.json())

        raise Exception('api exception', response)

    def post_order(self, order: Order) -> int:
        response = requests.post(f'{self.__base_url}/orders', order.json())
        return response.json()['id']
```

```python
class OrderApiGatewayDummy(IOrderApiGateway):

    def get_order(self, order_id: int) -> Optional[Order]:
        pass

    def post_order(self, order: Order) -> int:
        pass

    def __next_id(self):
        pass
```

Test Mock

- Útfæra sama interface
- Eru pre-programmed af test-unum
  - Test stjórna output úr föllum fyrir einhver inntök
  - Test stjórna hvort föll kasta villum eða ekki
  - Test stjórnað nákvæmlega hvað fall gerir
- Hafa *spy* virkni
  - Hægt að sannreyna fallaköll með ákveðnum inntökum og hversu oft
- Gott fyrir behaviour verification
  - Getur staðfest nákvæmlega hvað er kallað á
- Grá lína á milli Mocks og stub/dummy/spy
  - Mock hlutir, eins og þið munuð koma til með að þekkja þá, geta gert flest allt það sem mocks, spies og dummies gera
  - Flestir tala um Mocks sem yfirheiti fyrir test doubles
  - Í strangri merkingu er Mock hugtakið bara fyrir behvaiour verification þ.e.a.s að Mock hlutur veit hvaða föll ættu að vera keyrð og með hvaða parametrum
- Sjá Mock dæmi hér



Test-Driven-Development (TDD)

- *„Test-Driven Development (TDD) is a technique for building software that guides software development by writing tests.“*

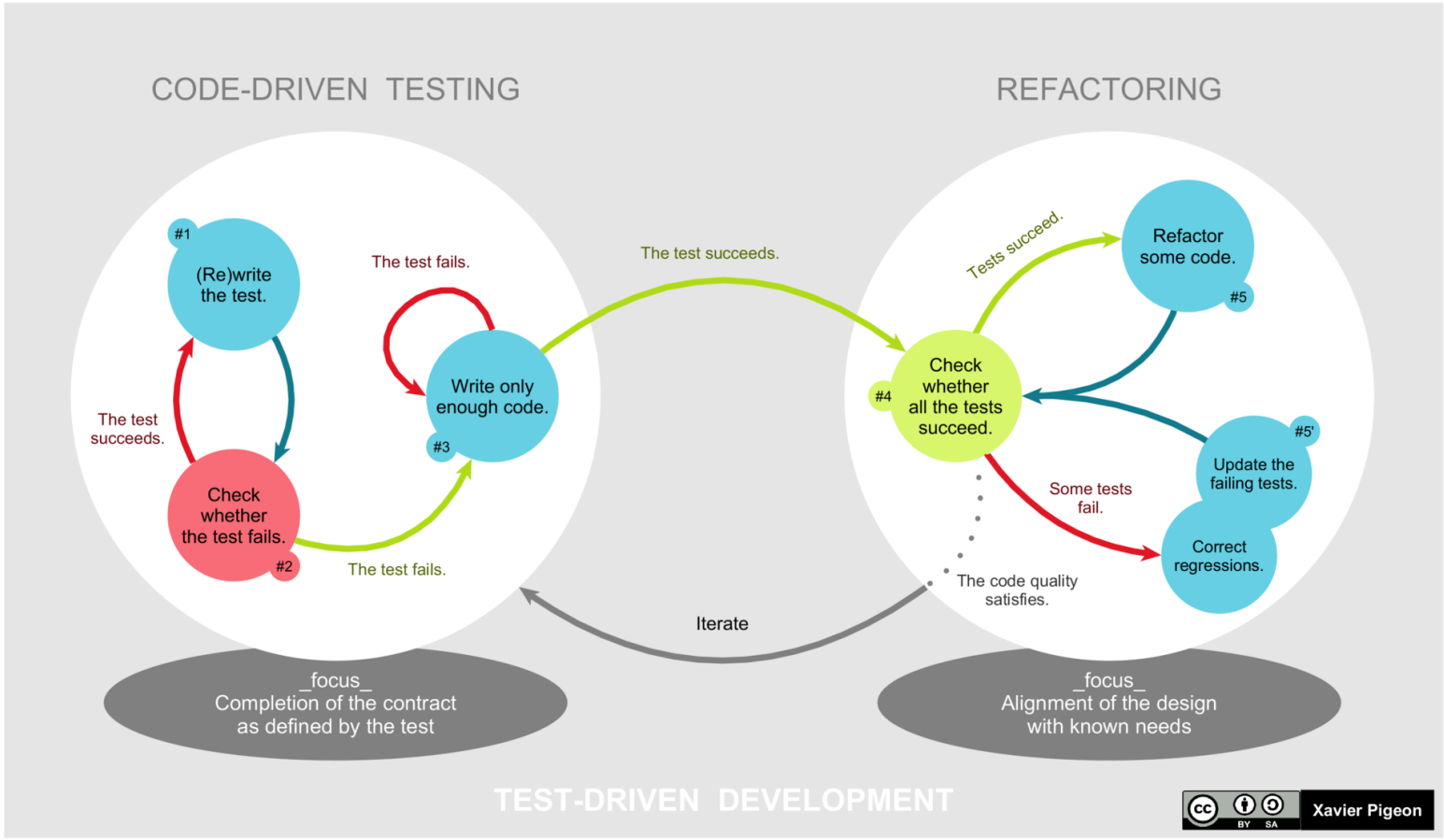
- Test Driven Development gengur út á að skrifa testin fyrir einhverja virkni áður en við skrifum kóðann fyrir þá virkni.

TDD Skrefin

1. Bæta við test fyrir virkni
   - Að bæta við nýrri virknibyrjar á því að búa til test fyrir þá virkni
2. Keyra test-in
   - Test-in ættu öll að fail-a fyrir þennan nýja feature
   - Önnur test ættu að vera passed
   - Sýnir að testin eru ekki gölluð og fái alltaf passed
   - Sýnir að kóða vantar svo test-in fái passed
3. Skrifa einfaldan kóða sem fær test-in til að gefa passed
   1. Ljótur kóði og harðkóðun er leyfileg
   2. Aðal málið að fá testin til að gefa okkur passed
4. Öll test ættu að fá passed núna
   - Ef einhver test fail-a þá þarf að laga kóðann
5. Refactor-a kóðann til að gera hann fallegri og betri
   - Eftir hvert refactor eru test-in keyrð til að vera viss um að breytingin braut ekki neitt
   - Hringur: Refactor -> Test -> Refactor -> Test ...

TDD Kostir

- Leið til að fá góð test fyrir allt kerfið
- Neyðir mann til að hugsa um interface-ið first
- Neyðir mann til að hugsa um kröfur **áður** en forritun hefst
- Neyðir mann til að hugsa um edge cases fyrst


CODE-DRIVEN TESTING
REFACTORING
#1
(Re)write the test.
The test fails.
The test succeeds.
Write only enough code.
#3
The test succeeds.
Check whether the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
#4
Check whether all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs
TEST-DRIVEN DEVELOPMENT
Xavier Pigeon

Best practices

- Unit test ættu að vera lítil og independent frá hvort öðru
  - Hver test ætti bara að test-a eitt unit í einu
  - Ætti ekki að skipta máli í hvaða röð unit test eru keyrð
  - Nota Mocks, Stubs etc. til að test-a unit independently
- Unit Test ættu að vera einföld
  - Ætti bara að testa eitt unit
  - Ætti bara að test-a einn hlut í því unit-i
  - Ættu að vera lítil
  - Yfirleitt bara hafa eitt “Assert” fyrir hvert test
- Unit Test ættu að vera læsileg
  - Ættu að segja sögu og document-a
  - Test eru líka kóði -> ættu að vera clean
- Unit Test ættu að vera áreiðanleg
  Ættu alltaf að fá “passed”
- Unit Test ættu að vera hröð
  Stór kerfi munu hafa þúsundir unit-testa
  Viljum testa kerfið fljótt og oft
- Nota skýrar nafnavenjur
  - Nafn á test föllum ættu að vera löng og lýsandi
  - Nota SUT breytunafn (SUT = System-Under-Test)
- Nota Arrange, Act, Assert strúktur
  - Arrange -> setur upp þau gögn, dependency og SUT
  - Act -> Keyrir kóðann sem þú ert test-a
  - Assert -> Athugar hvort að test-ið *passed*
- Truly Unit
  - Ætti ekki að access-a external resources
- CI/CD
  - Ættu að vera keyrð í CI/CD pipeline
- TDD

Pytest

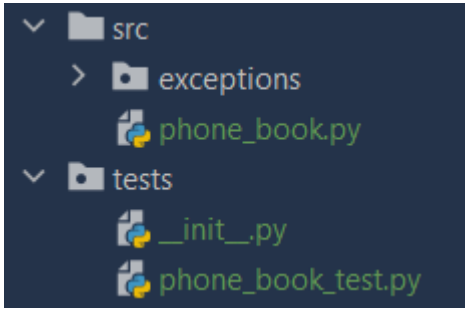
- Python testing framework sem við munum nota í þessum áfanga -> https://docs.pytest.org/en/6.2.x/
- Testing Framework hjálpa við ýmsa hluti:
  - Halda utan um strúktúrinn á test-unum
  - Setja upp test-in („set-up“)
  - „tear-down“ á testum
  - Keyra test-in
  - Gefa skýr output á þeim test-um sem failuðu og af hverju
  - Geta generate-að testing reports (t.d. Code coverage report)
  - Geta profile-að test-in (t.d. performance profiling)

Sýnidæmi

- Gerum Phonebook klasa
  - Phonebook á að hafa Add fall sem bætir phonenumber og nafni á einstakling sem á símanúmerið við eitthvað collection í phonebook hlutnum
    - Ef nafnið er nú þegar til þá yfirskrifast símanúmerið fyrir það nafn
    - Ef símanúmer er ekki gilt þá kastast villa af taginu InvalidNumberException
      - Símanúmer er valid ef það er 7 stafir og inniheldur eingöngu tölustafi
  - Phonebook á að hafa lookup fall sem tekur við nafni og skilar símanúmeri fyrir það nafn
    - Ef ekkert nafn er fundið þá ætti fallið að kasta KeyError

src/phone_book.py

tests/phone_book_test.py




```python
from src.exceptions.invalid_number_exception import InvalidNumberException


class PhoneBook:
    def __init__(self):
        self.__phone_numbers = {}

    def add(self, name: str, phone_number: str) -> None:
        if self.__is_phone_number_valid(phone_number):
            self.__phone_numbers[name] = phone_number
        else:
            raise InvalidNumberException()

    def __is_phone_number_valid(self, phone_number: str) -> bool:
        return phone_number and phone_number.isdecimal() and len(phone_number) == 7

    def lookup(self, name: str) -> str:
        return self.__phone_numbers[name]
```

```python
import pytest

from src.exceptions.invalid_number_exception import InvalidNumberException
from src.phone_book import PhoneBook


@pytest.fixture()
def phone_book():
    return PhoneBook()


def test_that_add_adds_valid_number_to_phonebook(phone_book):
    # Act
    phone_book.add('bob', '1234567')
    phone_number = phone_book.lookup('bob')

    # Assert
    assert phone_number == '1234567'


def test_that_add_doesnt_add_double_entry(phone_book):
    # Act
    phone_book.add('bob', '1234567')
    phone_book.add('bob', '7654321')
    phone_number = phone_book.lookup('bob')

    # Assert
    assert phone_number == '7654321'


@pytest.mark.parametrize('phone_number', ['1', '', None, 'abc', 'abcdefg', '12345678'])
def test_that_add_throws_invalid_number_exception_if_phone_number_is_invalid(phone_book, phone_number: str):
    # Arrange
    phone_book = PhoneBook()

    # Act Assert
    with pytest.raises(InvalidNumberException) as exception:
        phone_book.add('bob', phone_number)


def test_that_lookup_throws_key_error_when_nothing_is_found():
    # Arrange
    phone_book = PhoneBook()

    # Act Assert
    phone_book.add('bob', '7654321')
    with pytest.raises(KeyError) as exception:
        phone_book.lookup('bobby')
```

Pytest Code Coverage

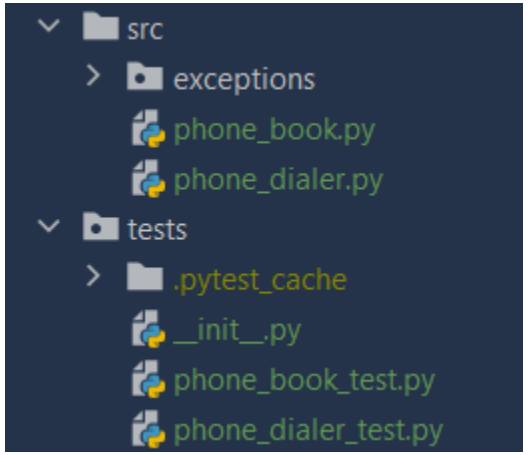
- Lýsing
  - Code Coverag er mælieining á hversu vel test-in cover-a kóðann og hversu vel test-in test-a kóðann

- Installation
  - `pip install coverage`
  - `pip install pytest-cov`

- Statement Coverage
  - Mælir prósentu af statement-unum í kóðanum sem test-in execute-a
  - Mjög basic -> (nr of executed statements) / (nr of statements)
  - Default fyrir pytest-cov
  - Terminal output
    - `pytest --cov-report term --cov=src`
  - Interactive html
    - `pytest --cov-report html:cov_html --cov=src`

- Önnur Code Coverage:
  - Branch Coverage
  - Decision Coverage
  - FSM Coverage

Mock dæmi




```python
class PhoneDialer:
    def __init__(self, phone_book: PhoneBook):
        self.__phone_book = phone_book

    def call_person(self, name: str):
        try:
            phone_number = self.__phone_book.lookup(name)
            print(f'calling {phone_number}')
        except KeyError:
            print(f'oh no, {name} does not exist')
```

```python
def test_that_call_person_dials_persons_phone_number(capsys):
    # Arrange
    phone_book = Mock(PhoneBook)
    phone_book.lookup.return_value = '1234567'

    sut = PhoneDialer(phone_book)

    # Act
    sut.call_person('bob')

    # Assert
    captured = capsys.readouterr()
    assert captured.out == 'calling 1234567\n'


def test_that_call_person_prints_correct_message_when_person_is_not_in_phonebook(capsys):
    # Arrange
    phone_book = Mock(PhoneBook)
    phone_book.lookup.side_effect = KeyError()

    sut = PhoneDialer(phone_book)

    # Act
    sut.call_person('bob')

    # Assert
    captured = capsys.readouterr()
    assert captured.out == 'oh no, bob does not exist\n'


def test_that_lookup_is_only_called_once():
    # Arrange
    phone_book = Mock(PhoneBook)

    sut = PhoneDialer(phone_book)

    # Act
    sut.call_person('bob')

    # Assert
    phone_book.lookup.assert_called_once_with('bob')
```