Testing
An Introduction to Software Testing
Gholamhossein Tavasoli @ ZNU
Who Am I?

- Gholamhossein Tavasoli
- PhD Student in Artificial Intelligence
- System Developer for 8 years
  - High Performance IDS / IPS
  - Industrial IDS
- Security (Semi) Expert, Member of ZNU Cert for 3 years
Type of Bugs

- Security
- Regression
- Business Logic
- Accessibility
- Performance
- User Interface
- Integration
- Internationalization
- Scaling
Some Famous Software Failures

- Heartbleed
  - Heartbeat Extension of TLS is used to keep Datagram TLS sessions open.
  - Simple request and response scheme:
    - "send me \textit{back the following (padded) string which is n bytes long.}"
  - An attacker just had to request a long string, while telling it is short.
  - Other party responded with short string, then leaked potentially confidential data.
- Video (https://youtu.be/WgrBrPW_Zn4)

heartbleed.com
Some Famous Software Failures

- Ariane-5
  - Inertial navigation software taken from Ariane-4. **Untested**.
  - All other systems thoroughly tested component-by-component.
  - Ariane-5 had a different trajectory than Ariane 4.
  - Converting 64-bit floating-point data into 16-bit unsigned integer values. → **Arithmetic overflow**.
  - There was an exception handler for that. **It had been disabled**.
  - Not even 40 seconds after launch, Ariane-5 literally self-destructed. Successfully.

- **Video** (https://youtu.be/WgrBrPW_Zn4)
Why Testing?

- Software contains defects. The hide.
- Defects can cause software failures.
- Failures can cost money, and even be mortal!
- Testing assures the quality of the software.
- Testing accelerates software development.
- And much, much more.
Why Testing?

- Software contains defects. They hide.
- Defects can cause software failures.
- Failures can cost money, and even be mortal!
- Testing assures the quality of the software.
- Testing accelerates software development.
- And much, much more.
What is Testing?

The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

- Testing has to be planned.
- Testing costs easily up to 40% of a project's budget.
- Testing has to be performed in a reasonable way.
- Testing shall be independent and objective.
- Testing has to be managed.

Software testing is a fundamental part of professional software development!
Type of Tests

- Unit Testing
- Integration Testing
- E2E Testing
- Penetration Testing
- Performance Testing
- Regression Testing
- a11y Testing
- Stress Testing
- Fuzz Testing
- A/B Testing
- User Acceptance Testing
- Usability Testing
- i18n Testing
- Smoke Testing
Test Levels

- Unit Testing
  - Test individual units in isolation.
  - Find defects in software components (e.g., functions, classes).
  - Done by developers.
  - In general, **white-box** tests.
What is a Unit Test?
Test Levels

- Unit Testing
- Integration Testing
  - Test communication/interaction of units (i.e., their interfaces).
  - Maybe separate unit integration and system integration tests.
  - Done by developers.
  - Primarily **white-box** tests, but also **black-box** tests.
What is a Integration Test?

Integration Test:
Does the component integrate into the system?

- Select droplet
- Select droplet
- Droplet list populates in dropdown
- Response shows droplet-2 details when selected
Test Levels

- Unit Testing
- Integration Testing
- System Testing
  - Test complete, integrated system.
  - Evaluate system compliance with specified requirements.
  - Stress, performance, usability etc. testing.
  - Done by (external) testers.
  - In general, black-box tests. Additional white-box tests possible.
Test Levels

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
  - Test complete, integrated system.
  - Evaluate system compliance with specified acceptance criteria.
  - May be performed at various times during development.
  - Done by customers/users.
  - Only black-box tests
App

- Technologies
  - Python
App

- Technologies
  - Python
  - Flask
  - Flaskr
  - NGINX
  - uWSGI

- Environment
  - Ubuntu Server (VM)
  - PyCharm
Setup

`git clone https://github.com/gtavasoli/testing-workshop.git`

`cd testing-workshop`
App Structure

- Unit Tests
- Simple Flask Apps
- Flaskr
- JMeter Profiles
Unit Test

Unit test in python
Unit Testing

- Sample 1 – Calculator
  - Test operators
  - Floating point division
  - Division by zero
- Sample 2 – Employee
  - Setup & Teardown
  - Setup & Teardown Class
Test Double

- Dummy Object
- Test Stub
- Test Spy
- Mock Object
- Fake Object
Test Double

- **Dummy** → I do nothing at all but to fill parameter list
- **Stub** → canned Answers
- **Spy** → stubs + interaction recording (for late interaction expectations verification)
- **Mock** → stubs + expectations on interaction
- **Fake** → I seem real but no
What Are Mocks?

- The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing.
- Mocking makes unit testing easier!
- Verifies behavior (calls) to a method.
- A mock object simulates the behavior of real objects in controlled ways.
What Are Mocks?
ProblemsMocksSolve

- EliminatesdependenciesintheCUT(classunder test)
  - IsolateUniteTests

```python
def foo(x):
    y = bar(x)
    if y > 10:
        return x+y
    return x-y
```

Copyright: Based on “Mocking Strategies” – slides from 26 to end of mocking sample
Problems Mocks Solve

- Tests methods that have no return value

```python
def foo(x):
    if x > 10:
        bar(x)
    else:
        something_else(x)
```

How do we know that bar(x) has been called?
Problems Mocks Solve

- Tests error handling

```python
def foo(filename):
    try:
        return parse_large_file(filename)
    except MemoryError:
        return """"""
```

How do we generate this exception???
Problems Mocks Solve

- Eliminate dependency on database calls
  - Speed up testing!

- Reduce test complexity
  - Don’t have to write complex logic to handle behavior of methods not under test

- Don’t have to wait to implement other methods
Problem: Friend Finder App

- Create a method to return a new, random friendship candidate
  - Must not show the same person
  - Must not show someone the user has already visited.
Easy Method

```python
def get_next_person(user):
    person = get_random_person()
    while person in user['people_seen']:
        person = get_random_person()
    return person
```

“Surely no one could have seen EVERYONE in the database!!!”
Write a Unit Test...

```python
class TestNewPerson(unittest.TestCase):
    def test_new_person(self):
        # arrange
        user = {'people_seen': []}
        expected_person = 'Ali'

        # action
        actual_person = get_next_person(user)

        # assert
        self.assertEqual(actual_person, expected_person)
```
May be it Works!!!

Ran 1 test in 0.003s

OK
Process finished with exit code 0
May be don’t!

Ran 1 test in 0.004s

FAILED (failures=1)
Process finished with exit code 1
60% OF THE TIME

IT WORKS EVERY TIME
Easy Method

```python
def get_next_person(user):
    person = get_random_person()
    while person in user['people_seen']:
        person = get_random_person()
    return person
```

What if knew the result of `get_random_person()`???
MOCK

ALL THE DEPENDENCIES.
from unittest.mock import patch

def test_new_person_with_mock(self):
    with patch('get_next_person.get_random_person') as mocked_get_rp:
        # arrange
        user = {'people_seen': []}
        expected_person = 'Ali'

        mocked_get_rp.return_value = 'Ali'

        # action
        actual_person = get_next_person(user)

        # assert
        self.assertEqual(actual_person, expected_person)
But what if we call it multiple times???

def test_new_person_with_mock(self):
    with patch('get_next_person.get_random_person') as mocked_get_rp:
        # arrange
        user = {'people_seen': ['Ali', 'Diba']}
        expected_person = 'Zeynab'

        mocked_get_rp.side_effect = ['Ali', 'Diba', 'Zeynab']

        # action
        actual_person_1 = get_next_person(user)

        # assert
        self.assertEqual(actual_person_1, expected_person)
Problem: Matching in Friend Finder App

- When a user sends a friend request...
  - If the other user "likes" it:
    - Send them both a message with contact info
  - If the other user "dislikes" it:
    - Let the user down gently...
  - If the other user hasn’t evaluated yet:
    - Display the "give it time" message
Implementation

def evaluate(person1, person2):
    if person1 in person2['likes']:
        send_email(person1)
        send_email(person2)
    elif person1 in person2['dislike']:
        let_down_gently(person1)
    elif person1 not in person2['likes'] and person1 not in person2['dislike']:
        give_it_time(person1)

How do we test this??? No return values!!!
Behavior Verification

```python
def test_evaluate(self):
    with patch('evaluate.let_down_gently') as mock_let_down:
        # arrange
        person1 = 'Ali'
        person2 = {
            'likes': ['Bita', 'Mohammad'],
            'dislike': ['Ali']
        }

        #action
        evaluate(person1, person2)

        #assert
        self.assertEqual(mock_let_down.call_count, 1)
```
Mocking Exceptions
Simple JSON reader

```python
def get_json(filename):
    try:
        return json.loads(open(filename).read())
    except (IOError, ValueError):
        return ""
```

How do we test something like this???
Testing JSON reader

- Test parsing a valid file
- Test an IOError (i.e. file missing)
- Test a ValueError (i.e invalid JSON)
How do I test open()

- Let’s just create a sample file!
Can you even mock a builtin?

def test_get_valid_json(self):
    with patch('__builtin__.open') as mock_open:
        # arrange
        filename = "dose_not_exist.json"
        mock_open.return_value = '{"foo": "bar"}'

        # action
        actual_result = get_json(filename)

        # assert
        self.assertEqual({u'foo': u'bar'}, actual_result)
What? Why Not???

- open() returns a File object
- open(filename).read()
- So we really need to mock File.read()
  - But it’s an instance!!! Oh no!
- Have you tried solving it withMocks???
Mocks returning Mocks?

```python
class TestGetJson(unittest.TestCase):
    def test_get_valid_json(self):
        with patch('builtins.open') as mock_open:
            # arrange
            filename = "does_not_exit.json"

            mock_file = Mock()
            mock_file.read.return_value = '{"foo":"bar"}'
            mock_open.return_value = mock_file

            # action
            actual_result = get_json(filename)

            # assert
            self.assertEqual({u'foo': u'bar'}, actual_result)
```
What about IOError?

def test_get_json_ioerror(self):
    with patch('builtins.open') as mock_open:
        # arrange
        filename = "does_not_exit.json"
        mock_open.side_effect = IOError

        # action
        actual_result = get_json(filename)

        # assert
        self.assertEqual('', actual_result)
What about `ValueError`?

```python
def test_get_json_ValueError(self):
    with patch('builtins.open') as mock_open, \
        patch('json.loads') as mock_loads:
        # arrange
        filename = "does_not_exit.json"

        mock_file = Mock()
        mock_file.read.return_value = '{"foo":"bar"}'
        mock_open.return_value = mock_file
        mock_loads.side_effect = ValueError

        # action
        actual_result = get_json(filename)

        # assert
        self.assertEqual('', actual_result)
```
Warning

- A warning about over-isolating
  - Many people fall for relentless isolation
  - Magic tool for helping in isolating parts
- Mocking is about object communication and interface discovery
  - Using it for isolation is a misuse
  - Specially from 3rd party code
- General rule of thumb: "do not mock what you don’t own"
Warning

- Avoid over-mocking
  - Duplicating implementation code in the tests
  - Keep production and test code in sync
- We should refrain from this and try as much as possible to use real collaborators when possible.

*When your tests use the same collaborators as your application, they always break when they should. The value of this cannot be underestimated*

— Sandi Metz & Katrina Owen
Testing/Coding is so hard!!
راه دیگری وجود ندارد.
Type of Software Testing

Types of Testing

Functional
- Unit Testing
- Integration Testing
- Smoke / Sanity
- User Acceptance
- Localization
- Globalization
- Interoperability
- So on ...

Non-Functional
- Performance
- Endurance
- Load
- Volume
- Scalability
- Usability
- So on ...

Maintenance
- Regression
- Maintenance
Non-functional Testing

- Performance Testing
  - To identify any bottlenecks or performance issues rather than finding bugs in a software
  - Speed
  - Capacity
  - Stability
  - Scalability
Non-functional Testing

- Performance Testing
- Load Testing
  - The process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data
- Tools
  - Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.
Non-functional Testing

- Performance Testing
- Load Testing
- Stress Testing
  - Includes testing the behavior of a software under *abnormal conditions*
  - Applying the load to the system and taking over the resources used by the software to identify the breaking point
Non-functional Testing

- Performance Testing
- Load Testing
- Stress Testing
  - Shutdown or restart of network ports randomly
  - Turning the database on or off
  - Running different processes that consume resources such as CPU, memory, server, etc.
Non-functional Testing

- Performance Testing
- Security Testing
  - Confidentiality
  - Integrity
  - Authentication
  - Availability
  - Authorization
  - ...

...
Performance Testing

- Methodologies
  - Real User Monitoring - RUM
  - Synthetic Testing

- The Recipe
  - Step 1 – Collect Data
  - Step 2 – Analyze
  - Step 3 – Experiment
  - Step 4 – GoTo 1
Targets in Performance Testing

- Things we measure
  - Availability or uptime
  - Concurrency
  - Throughput
  - Response time
  - Network utilization
  - Server utilization
Tips

- Beware of failures
- Be aware of caching
- Automate bookkeeping
Tutorial 1 – Simple Test

- Step 1 – Start JMeter
- Step 2 – Create a TestPlan
- Step 3 – Create a Thread Group (Users)
- Step 4 – Add Sampler (HTTP)
- Step 5 – Add Listeners
- Step 6 – To Run the Test
Tutorial 2 – Using Assertions

- Response Assertion
- Duration Assertion
- Size Assertion
- HTML Assertion
Tutorial 3 – Flaskr

- Simple Microblog With Flask
- Parametric Request
Tutorial 4 – JMeter Plugins

- https://jmeter-plugins.org/
Tutorial 4 – JMeter Plugins

- JMeter Plugin: Response Times Percentiles / Distribution
- Tail Latency

"consider a system where each server typically responds in 10ms but with a 99th percentile latency of one second. If a user request is handled on just one such server, one user request in 100 will be slow (one second)."

The Tail at Scale – Dean and Barroso 2013
Tutorial 4 – JMeter Plugins

- What happens when there is not just a single server involved in servicing a request, but 100 (fan-out query, contribution of a large number of microservices)
If a user request must collect responses from 100 such servers in parallel, then 63% of user requests will take more than one second. Even for services with only one in 10,000 requests experiencing more than one-second latencies at the single-server level, a service with 2,000 such servers will see almost one in five user requests taking more than one second.
Tutorial 4 – JMeter Plugins

- Tail Latency
- Root Causes
  - Shared resources (memory, CPU cores, CPU caches, shared file system)
  - Background tasks running on the server
  - Maintenance activities (GC, Log rotation)
  - Queues in general

https://blog.acolyer.org/2015/01/15/the-tail-at-scale/