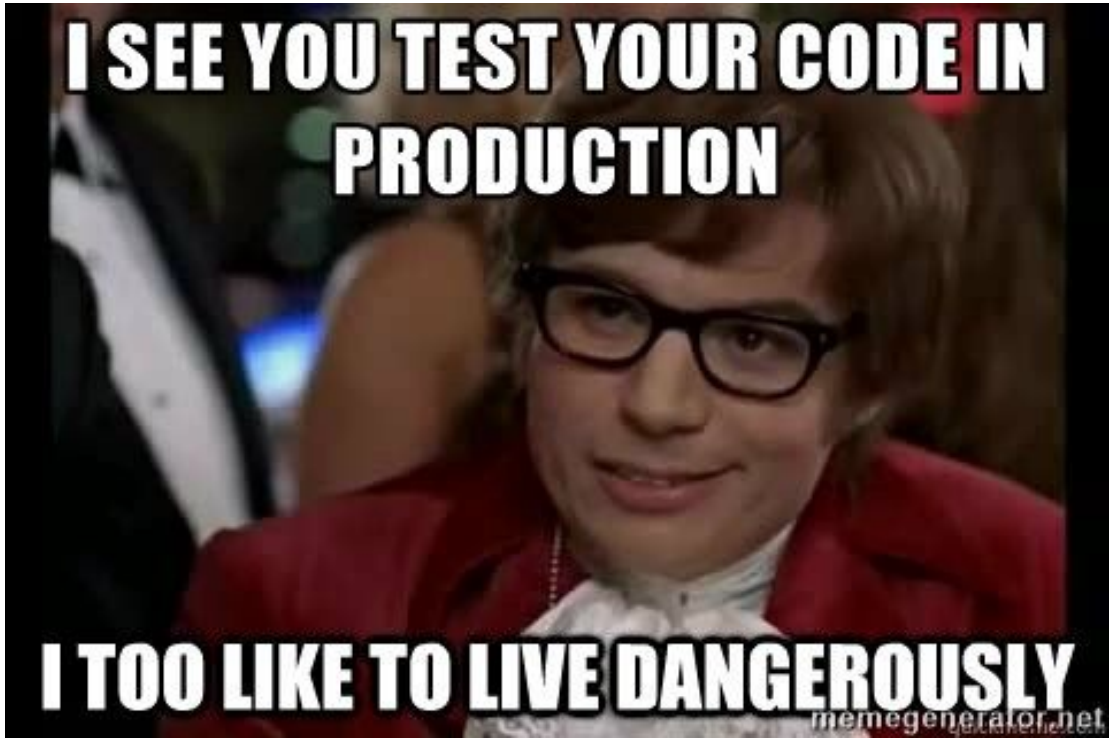
Automated Testing

Overview

- Why should I test my code?
- What is automated testing?
- Some benefits of automated testing
- Very high level overview of automated testing

Why We Should I Test My Code?



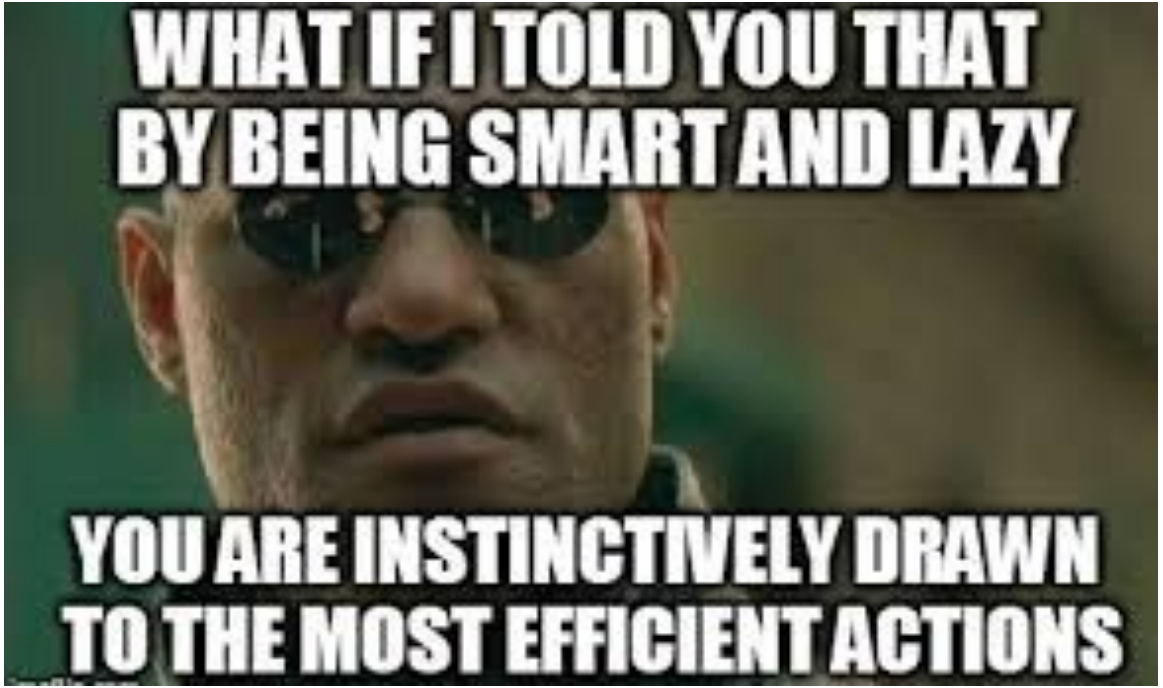
Tests Are Like Experiments

Experiment	Tests
Setup your experiment (environment, apparatus, etc)	Put your system under test in a known state
Run the experiment	Execute the test on your system
Analyse the results ● Did you get what was expected?	Validate that your result is what you expected
Repeat experiment, perhaps with different parameters	Repeat, perhaps with different states

But What are Automated Tests?

Repeatable tests that have been codified that can be executed on or by a computer

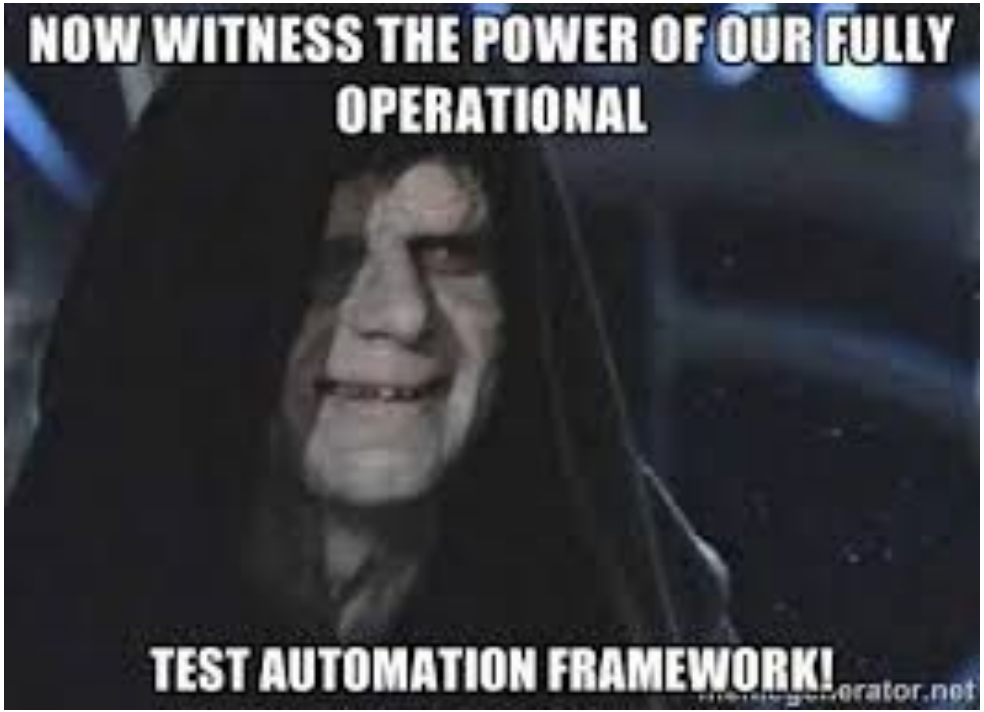
Humans Are Lazy



Automation is Smart!

Automation Helps With Productivity

Testing And Automation Together!



Why Should 'I' Automate Tests?

- Humans are fallible, even on repeatable tasks. Having automated tests mean that your code is tested regularly and the same way. If tests start to fail something else has gone wrong (your code, your data, your environment, ...).
- Every time you modify your code or system you are potentially adding 'bugs', automated tests help make sure you keep on top of them.
- Unless it's your own personal project, someone else is going to have to read, use, and maintain your code. They may even need to test your code.
- 'Future You' is another person, your tests will help future you maintain your code.

Benefits of Test Automation

- Helps to document expected behaviour of your code.
- Earlier Detection of Defects.
- Faster feedback about is something has gone wrong.
- Testing Efficiency:
  - Large projects set a lot of time set aside for testing. Automated tests allows more efficient use of people's time.
  - Testers get to do more exploratory testing rather than labourious manual tasks. The bug fixes can have an automated test to verify fix.
- Automated Tests can be run anytime and anywhere.
- Tests can be run in parallel.
- Manual testing is important! It helps finds problems within **new** code.

Personal Case Study

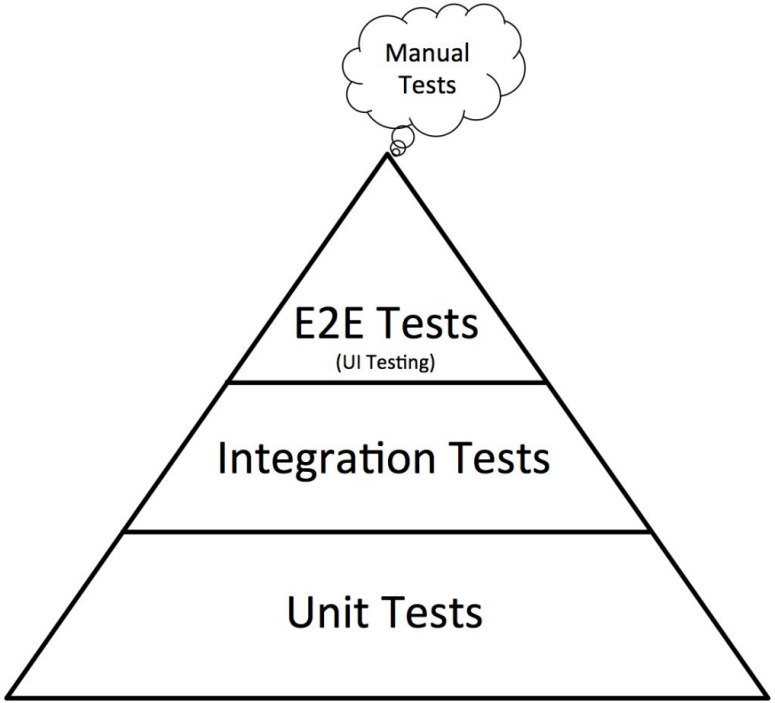
Personal Case Study

- Team built 3 green field applications
- Used Test Driven Development as part of the process to build the apps
- Only ever had 1 major bug within production
- Got to the point we could deploy anytime of the day
- Other teams had a fortnightly release cycle including 2-3 day testing window

Automated Testing 101

- Automated Testing is a big area and there's a lot to cover
- We need to start somewhere, so I'm just going to scratch the surface
- There is a lot of content on automated testing online (O'Reilly Online great source of eBooks - https://learning.oreilly.com/ - Use your SIMS login)

Testing Pyramid



Source: Ministry of Testing **Credit:** Daniel Knott

Unit Testing

- Very small tests, testing a ‘unit’ of work: a method, or even a class.
- You’re in control of what you specifically want to test
- You’re in control of the priors and you know what the posterior should be
- Use **fake** data, but try to be representative of the data you expect at runtime
- Use **minimalistic** input files. Should only have a few records (~ 1-10 records)
- Avoid integration - no database, no HTTP, no sockets, etc

Unit Testing (cont)

1. Put the code you want to test in a known state (i.e. state your priors)
2. Execute your code (run your experiment).
3. Verify the result returned from you code matches what you expect (check results of experiment matches what was expected).

```
def test_set_value(self):
    self.cei.set_value('abc', 'foo')

    result = self.cei.get_value('abc')

    assert result == 'foo'
```

Functional/Integration Testing

- Unit Testing on steroids
  - Have system in known state
  - You are still in control of inputs and expectations
- A test still only tests one thing - one requirement at a time
- Should integrate with the likes of databases.
- Use more larger and realistic data (5 records in a file vs 1000 records)
- These document your **business** requirements
- Tests should more human readable, as they will be your documentation
- Like Unit Tests these should be quick to run. Get quick feedback

Test Doubles

- Think of these as your ‘stunt doubles’ for your code
- They stand in for parts of your code that you don’t want to test
  - This might code you already have tested
  - Might be a 3rd party library
- Mocks vs Stubs
  - Mocks are used to test behaviour, very common in the Java ecosystem. You can test that it was called, they can be programmatically told what to return when called with specific values
  - Stub are used to provide canned responses.

Code Coverage

- Measures how much of your code is tested
- Code coverage tools like **pytest-cov** can report on what lines of code are tested and what's not.

```
Name                                          Stmts   Miss  Cover
-----------------------------------------------------------------
ce_interface/__init__.py                          4      0   100%
ce_interface/ce_interface.py                     43      4    91%
ce_interface/configuration.py                    44     15    66%
ce_interface/shared_mem_obj_repository.py        27      2    93%
ce_interface/value_store.py                      27      0   100%
setup.py                                          4      4     0%
tests/test_ce_interface.py                       34      0   100%
tests/test_configuration.py                      14      0   100%
tests/test_value_store.py                        39      0   100%
-----------------------------------------------------------------
TOTAL                                           236     25    89%
```

Code Coverage (cont)

- Old School says you should get **100%** but this can cause brittleness, a small change in code could require a lot of changes in tests.
- Try to aim for **100%** but if you get less don't worry but do set a hard lower limit
- You should also check **branch coverage**. You coverage report may say something is covered by **AND** and **OR** conditionals might not be exercised

```
In [ ]: if a or b:
            # do something

        if a and b:
            # do something
```

Somethings to think about

- Remember - If you can repeat a manual test then you can automate it
- Get in the habit of automating your testing
- Keep your tests minimal (Fail fast, fail hard)
- Adding tests will help you (and others) know what the code is doing even years after code was developed

An Important Point

Even a few
automated tests are
better than no tests

References & Resources

Top 10 Benefits of Automated Testing
https://saucelabs.com/blog/top-10-benefits-of-automated-testing

Ministry of Testing - Testing Pyramid
https://dojo.ministryoftesting.com/dojo/lessons/the-mobile-test-pyramid

Martin Fowler - The Practical Test Pyramid
https://martinfowler.com/articles/practical-test-pyramid.html

O'Reilly Online Learning
https://www.oreilly.com/sign-in.html (Use your SUT email for login, then click on 'Sign In With Single Sign On')

O'Reilly Online Learning

https://www.oreilly.com/sign-in.html