Testing

```
section .data
code            db "COMP1929",
name            db "Software Engineering",
credits         db 15
module_leader   db "Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA",
```


Download as a PDF

Why do we Test?

- Ensures a high quality of code.
- Catches bugs early.
- Ensures we are ‘building the product right’.

- If tests pass, the behaviour of the program is what you expect.
- If tests fail, then your program is likely not performing as it should.

- If tests fail then one of two things has happened:
  - Programming errors: You have made an error in the logic of your code (perhaps an out fo bounds error)
  - Understanding errors: You have not understood the requirements, or overlooked definition in the problem

Types of Testing

[1] Feature Tests

[2] Functional Testing

[3] Performace and Load Testing

[4] Security Testing

[5] Test Driven Development

[6] Unit Testing

[7] User Testing

Feature Tests

- When units become a feature, you may want to write tests to check the functionality of the entire feature to ensure that:
- The units are interacting with each other
- The feature acts as it should

Functional Testing

- Functional testing is a way of testing all of the ‘moving parts’ of a software product.
- You should aim to test the entire codebase
- For enterprise level systems this could me test suites of thousands of tests.
- You will be familiar with unit testing which is a type of test that we employ to test the smallest parts (or units) of a program
- We also use feature tests, system testing and release testing.

Performance and Load Testing

- If you are developing an API or system that is going to receive millions of calls per day, the last thing you want is for your software to fail (Pokémon Go makers call for calm as servers crash across Europe and US)

- Many services will have service level agreements (SLA's) such as guaranteeing 99.999% uptime

- During development you need to ensure that performance will continue as expected and that your system is capable of handling the transactional load required.

Security Testing

- Security testing is, as it sounds, testing to find the vulnerabilities in the software product being developed. Pen testing teams will usually be outsourced to test the vulnerabilities in your system.

- Unhashed passwords

- Unsanitised text inputs (`“Enter your surname:” DROP TABLE customer_list;`)

- DDoS attacks

- HTTP session cookies are visible

- *Pressing the back button and then ‘show password’*

Test Driven Development

- You should have many test cases for this one method/ function.
- In fact most code will be written after the test
- We call this test-driven development (TDD)

Test Driven Development

- We also use the Red Green Refactor cycle
  - We write a (failing) test where the test cases will pass when we build the right function
  - We write our code until the test is passing
  - We refactor our code

center

TDD vs BDD

- TDD is more developer-centric, revolving around code correctness, with programming language-specific frameworks.

- On the other hand, BDD is more user-centric, revolves around system behavior, and promotes collaboration between relevant stakeholders with a domain-specific language.

- BDD starts with analyzing the desired behavior that developers want to create. After that, they'll express the desired behavior using the Gherkin syntax, which consists of **Given - When - Then - And** statements. These statements show developers how to develop the code that fulfills the behaviors described.

BDD

- Gherkin is a plain-text language with a simple structure.
- It is designed to be easy to learn by non-programmers
- Allows concise description of test scenarios and examples to illustrate business rules in most real-world domains.

```
Feature: Account Holder withdraws cash

Scenario: Account has sufficient funds
    Given The account balance is $100
      And the card is valid
      And the machine contains enough money
    When the Account Holder requests $20
    Then the ATM should dispense $20
      And the account balance should be $80
      And the card should be returned
```

Unit Testing

- We write unit tests to test units of the program
- The are usually functions or class methods
- Unit tests take a function, present it some input and test its output
- When writing unit tests it is a good idea to use the **Arrange-Action-Assert pattern**

Unit Testing

- Consider the `abs()` function in Python and `Abs()` in C#
- `abs()` / `Abs()` returns the absolute value of a number (the distance of the number from zero)
- Using **Arrange-Action-Assert** we can write a test to ensure the behaviour of `abs()` / `Abs()` is what we expect

```python
# Test abs
def test_abs()

    # Arrange (Setup the testing scenario, mock any data)
    value = 7.5

    # Action (call the function to be tested)
    result = abs(value)

    # Assert (Provide the expected outcome)
    assert result == 7.5
```

```csharp
[TestClass]
public class MathTests
{
    [TestMethod]
    public void TestAbs()
    {
        // Arrange (Setup the testing scenario, mock any data)
        double value = 7.5;

        // Action (call the function to be tested)
        double result = Math.Abs(value);

        // Assert (Provide the expected outcome)
        Assert.AreEqual(7.5, result);
    }
}
```

User Testing

- Usability testing
  - Can users learn to use the system quickly
  - Can the users complete tasks using the system without making errors?
- Utility testing
  - Can the users do what they need with the system?
- User interface testing
  - Are the users happy with the interface?

User Testing

- Alpha testing
  - Usually the job of UX Designers
  - Do the users actually want these features?
  - What have you missed?
  - Involve focus groups early on

- Beta testing
  - Far more users
  - Aware that some bugs may persist, but may be edge cases
  - Windows Release candidates, iOS betas etc.

System Tests and Release Tests

- When features make up a product or usable system, you will of course want to test that system and in turn when this becomes a release

- Think about the end of a sprint when you are ready to release your product to your users you will again want to test that the release functions on all platforms/ hardware etc.

```yaml
name: Upload Python Package

on:
  release:
    types: [published]

permissions:
  contents: read

jobs:
  release-build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.10"

      - name: Build release distributions
        run:  python -m pip install build && python -m build

      - name: Upload distributions
        uses: actions/upload-artifact@v4
        with:
          name: release-dists
          path: dist/

  pypi-publish:
    runs-on: ubuntu-latest
    needs:
      - release-build
    permissions:
      id-token: write

    environment:
      name: pypi
      url: https://pypi.org/project/pylings/${{ github.event.release.name }}

    steps:
      - name: Retrieve release distributions
        uses: actions/download-artifact@v4
        with:
          name: release-dists
          path: dist/

      - name: Publish release distributions to PyPI
        uses: pypa/gh-action-pypi-publish@release/v1
        with:
          packages-dir: dist/
```

Testing as part of the Pipeline

- Remember when we said last week that we ran our tests before we deployed our software?
  - What we really mean are our **functional** tests (Unit, Feature, System and Release tests)

- Things like user testing are undertaken ad-hoc and usually arranged by the Product Manager or Scrum Master

- In bigger organisations performance and load testing may undertaken by you infrastructure team

- Security testing should always involve a third party. Either you don’t have the expertise in your organisation or you will be biased and fail to test something important.