NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

A TASTE OF AGILE METHODS: KANBAN AND UNIT TESTS

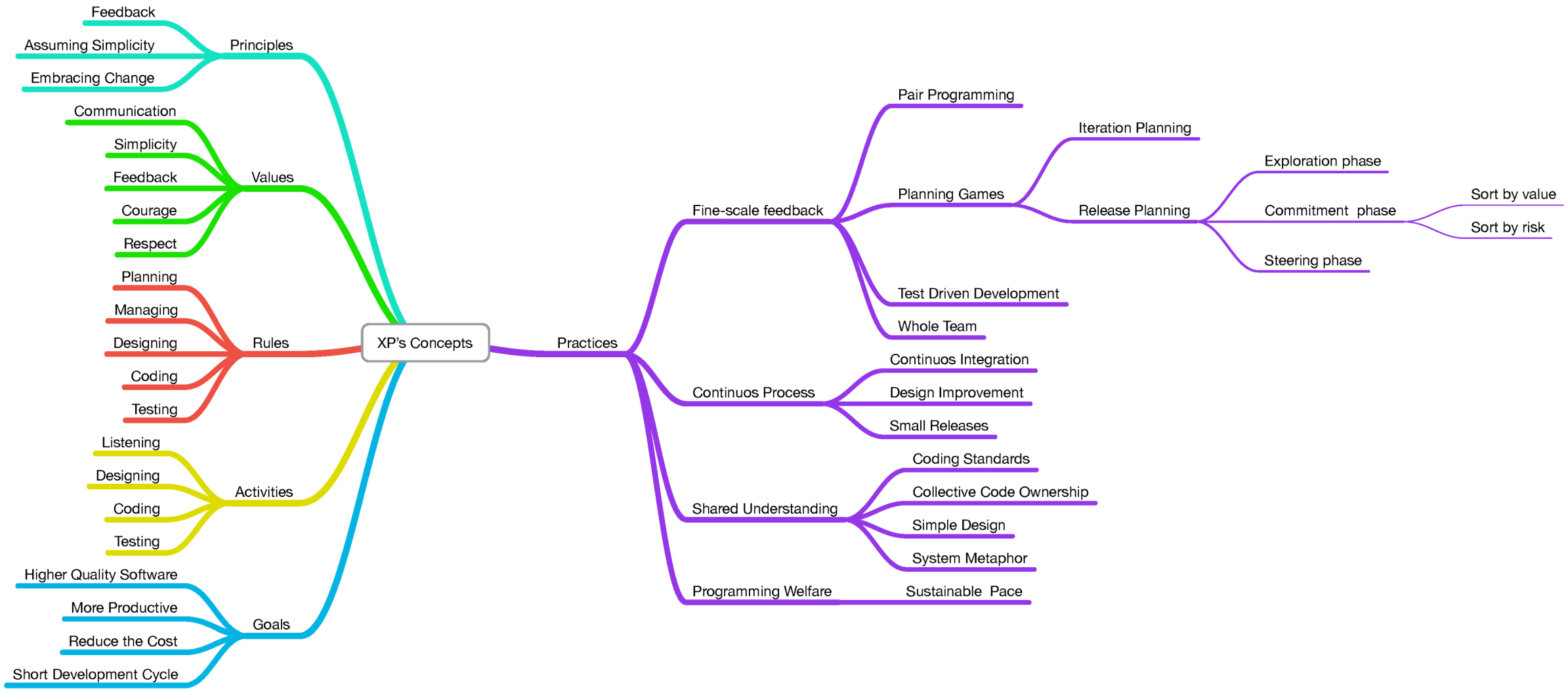
INF502

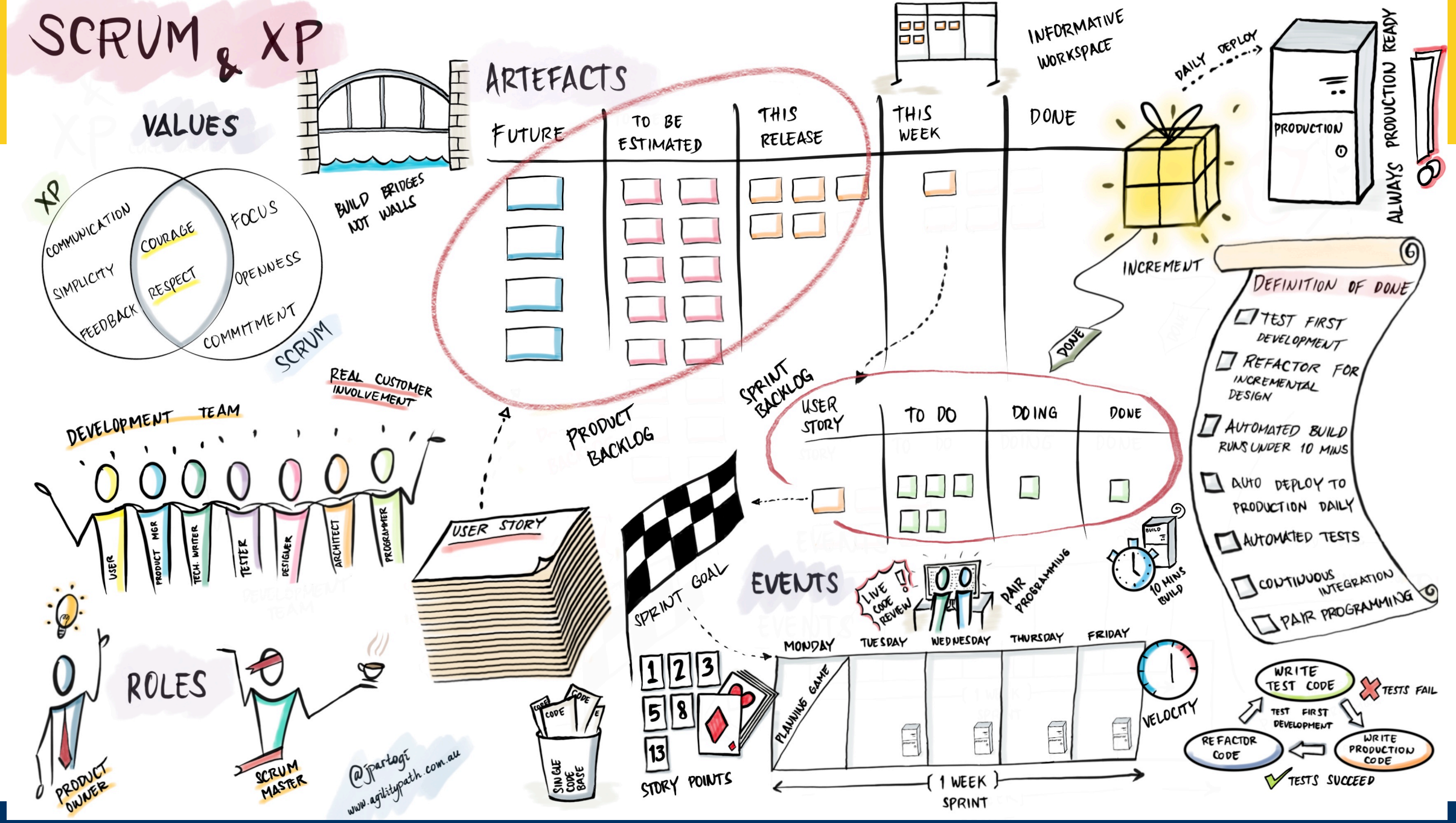
Igor Steinmacher/Igor Wiese

Manifesto for Agile Software Dev.

AGILE

- INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS
- WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTION
- CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION
- RESPONDING TO CHANGE OVER FOLLOWING A PLAN


XP's Concepts
Principles
Feedback
Assuming Simplicity
Embracing Change
Values
Communication
Simplicity
Feedback
Courage
Respect
Rules
Planning
Managing
Designing
Coding
Testing
Activities
Listening
Designing
Coding
Testing
Goals
Higher Quality Software
More Productive
Reduce the Cost
Short Development Cycle
Practices
Fine-scale feedback
Pair Programming
Planning Games
Iteration Planning
Release Planning
Exploration phase
Commitment phase
Sort by value
Sort by risk
Steering phase
Test Driven Development
Whole Team
Continuos Process
Continuos Integration
Design Improvement
Small Releases
Shared Understanding
Coding Standards
Collective Code Ownership
Simple Design
System Metaphor
Programming Welfare
Sustainable Pace



https://dzone.com/articles/scrum-and-extreme-programming-xp

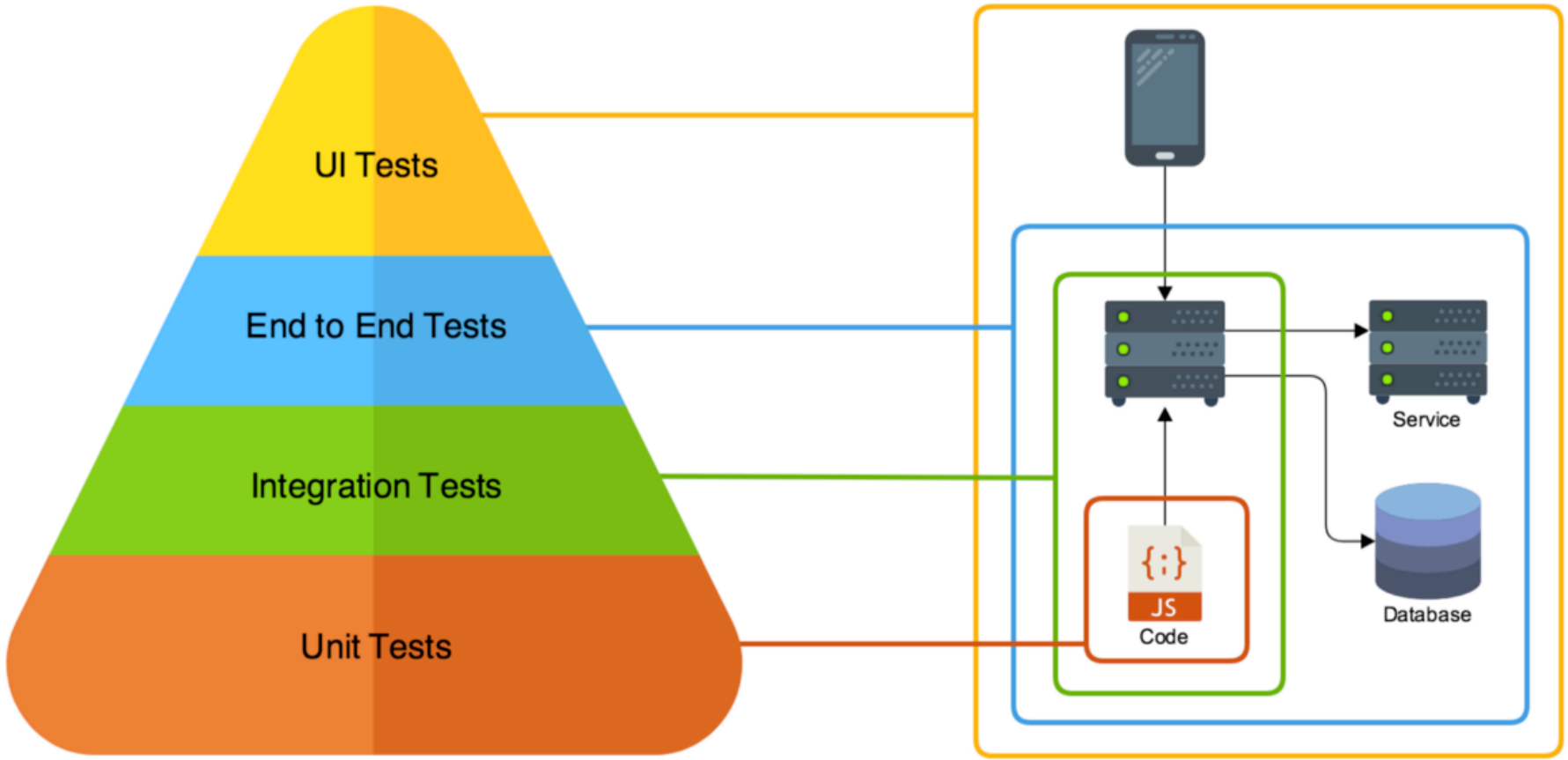
Our Focus

KANBAN

- Let’s use a tool

Tests

- We will talk about Unit Testing



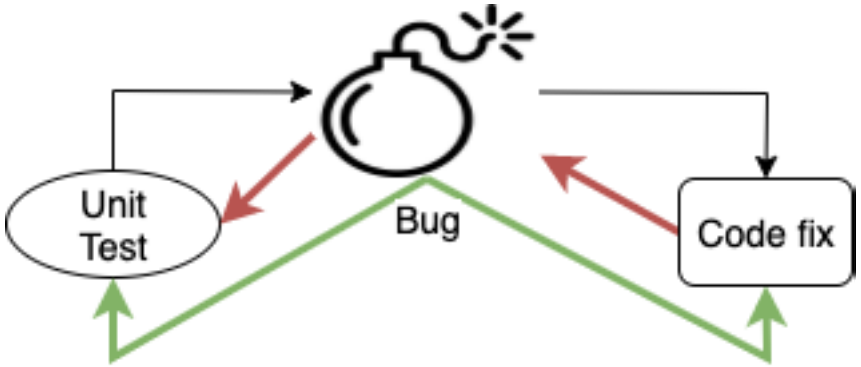
Unit Tests

- First level of software testing
- The smallest testable parts of a software are tested
- Validate each unit of the software
- Each test unit must be fully independent

Unit Tests: Why?

- Check if pieces are working after you change the system
  - Unit testing increases confidence in changing/ maintaining code.
- Point out defects while developing
- Enforces more reusable code → You need to go modular
- Debugging is easier (sometimes)
- Helps you finding where the issue is

Unit Test



Unit Test: How To?

- There are multiple modules we can use
- We will focus on creating Testcases using `unittest` (part of Python standard package)
  - **Import the module:**

```
import unittest
```

  - **Import the file/function(s) you are testing:**

```
import function from file
```

  - **Create a class that extends Testcase**

```
class myTest(unittest.TestCase):
    ...
```

  - **Convert the tests into methods**
    - Using assertions

Unit Test: Assert What? Who?

- Asserts: Methods we can call to evaluate our functions
  - **assertEqual()-** Tests that the two arguments are equal in value.
  - **assertNotEqual()-** Tests that the two arguments are unequal in value.
  - **assertTrue()-** Tests that the argument has a Boolean value of True.
  - **assertFalse()-** Tests that the argument has a Boolean value of False.
  - **assertIs()-** Tests that the arguments evaluate to the same object.
  - **assertIsNot()-** Tests that the arguments do not evaluate to the same object.
  - **assertIsNone()-** Tests that the argument evaluates to none.
  - **assertIsNotNone()-** Tests that the argument does not evaluate to none.
  - **assertIn()-** Tests that the first argument is in the second.
  - **assertNotIn()-** Tests that the first argument is not in the second.
  - **assertIsInstance()-** Tests that the first argument (object) is an instance of the second (class).
  - **assertRaises()-** Tests that Python raises an exception when we call the callable with positional/ keyword arguments we also passed to this method.
  - **More here:**

    https://kapeli.com/cheat_sheets/Python_unittest_Assertions.docset/Contents/Resources/Documents/index

Unit Tests: Basic!

```
import unittest class TestSum(unittest.TestCase):

def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

def test_sum_tuple(self):
        self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

if __name__ == '__main__':
        unittest.main()
```

```
.F
======================================================================
FAIL: test_sum_tuple (__main__.TestSum)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "basicTest.py", line 8, in test_sum_tuple
    self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")
AssertionError: 5 != 6 : Should be 6

----------------------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

UNIT TESTS: BASIC!

```
def is_prime(number):
        for element in range(number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

The smallest unit here is the **is_prime** function (**print_next_time** uses it)

Let's test it using something that we know
Is 5 prime?

Thanks
https://jeffknupp.com/blog/2013/12/09/improve-your-python-understanding-unit-testing/

Unit Tests: Basic!

```
def is_prime(number):
        for element in range(number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
   def test_is_five_prime(self):
      self.assertTrue(is_prime(5))
      #self.assertEquals(is_prime(5), True)

if __name__ == '__main__':
        unittest.main()
```

```
E
======================================================================
ERROR: test_is_five_prime (__main__.PrimesTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "PrimesTestCase.py", line 6, in test_is_five_prime
    self.assertTrue(is_prime(5))
  File "/Users/igorsteinmacher/Downloads/prime.py", line 3, in is_prime
    if (number % element == 0):
ZeroDivisionError: integer division or modulo by zero

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)
```

Unit Tests: Fixing…

```
def is_prime(number):
        for element in range(1, number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
   def test_is_five_prime(self):
      self.assertTrue(is_prime(5))

if __name__ == '__main__':
        unittest.main()
```

```
F
======================================================================
FAIL: test_is_five_prime (__main__.PrimesTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "PrimesTestCase.py", line 6, in test_is_five_prime
    self.assertTrue(is_prime(5))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.001s
```

Unit Tests: Fixing…

```
def is_prime(number):
        for element in range(2, number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
   def test_is_five_prime(self):
      self.assertTrue(is_prime(5))

if __name__ == '__main__':
        unittest.main()
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

WHAT TO TEST?

- Known cases (like we've done)

```
def is_prime(number):
        for element in range(2, number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    def test_is_five_prime(self):
        self.assertTrue(is_prime(5))
    def test_is_four_prime(self):
        self.assertFalse(is_prime(4))

if __name__ == '__main__':
        unittest.main()
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

WHAT TO TEST?

- Edge cases (ZERO!!!)

```
def is_prime(number):
        for element in range(2, number):
                if (number % element == 0):
                        return False
        return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    def test_is_five_prime(self):
        self.assertTrue(is_prime(5))
    def test_is_four_prime(self):
        self.assertFalse(is_prime(4))

    def test_is_zero_not_prime(self):
        self.assertFalse(is_prime(0))
```

```
..F
======================================================================
FAIL: test_is_zero_not_prime (__main__.PrimesTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "PrimesTestCase_5.py", line 10, in test_is_zero_not_prime
    self.assertFalse(is_prime(0))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
```

WHAT TO TEST?

```
def is_prime(number):
    if number in (0, 1):
        return False

    for element in range(2, number):
        if (number % element == 0):
            return False
    return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    def test_is_five_prime(self):
        self.assertTrue(is_prime(5))
    def test_is_four_prime(self):
        self.assertFalse(is_prime(4))

    def test_is_zero_not_prime(self):
        self.assertFalse(is_prime(0))
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

WHAT TO TEST?

```
def is_prime(number):
    if number in (0, 1):
        return False

    for element in range(2, number):
        if (number % element == 0):
            return False
    return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    ...
    def test_negative_number(self):
        self.assertFalse(is_prime(-1))
        self.assertFalse(is_prime(-2))
        self.assertFalse(is_prime(-3))
        self.assertFalse(is_prime(-4))
        self.assertFalse(is_prime(-5))
        self.assertFalse(is_prime(-6))
        self.assertFalse(is_prime(-7))
        self.assertFalse(is_prime(-8))
        self.assertFalse(is_prime(-9))
```

```
...F
======================================================================
FAIL: test_negative_number (__main__.PrimesTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "PrimesTestCase_6.py", line 12, in test_negative_number
    self.assertFalse(is_prime(-1))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 4 tests in 0.000s
```

WHAT TO TEST?

```
def is_prime(number):
    if number <= 1:
        return False

    for element in range(2, number):
        if (number % element == 0):
            return False
    return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    ...
    def test_negative_number(self):
        self.assertFalse(is_prime(-1))
        self.assertFalse(is_prime(-2))
        self.assertFalse(is_prime(-3))
        self.assertFalse(is_prime(-4))
        self.assertFalse(is_prime(-5))
        self.assertFalse(is_prime(-6))
        self.assertFalse(is_prime(-7))
        self.assertFalse(is_prime(-8))
        self.assertFalse(is_prime(-9))
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

WHAT TO TEST?

```
def is_prime(number):
    if number <= 1:
        return False

    for element in range(2, number):
        if (number % element == 0):
            return False
    return True


def print_next_prime(number):
        index = number
        while True:
                index += 1
                if is_prime(index):
                        print(index)
```

```
import unittest
from prime import is_prime

class PrimesTestCase(unittest.TestCase):
    ...
    def test_TypeError(self):
        self.assertRaises(TypeError, is_prime, "a")
```

```
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

WHAT TO TEST?

- Create unit tests for calculator below:

```
class Calculator:
    def add(x, y):
        return x + y

    def subtract(x, y):
        return x - y

    def multiply(x, y):
        return x * y

    def divide(x, y):
        return x / y

    def pow(x, y):
        res = 1
        for i in range(2,y):
            res = res*i
```