Spending less time bug fixing by spending more time unit testing

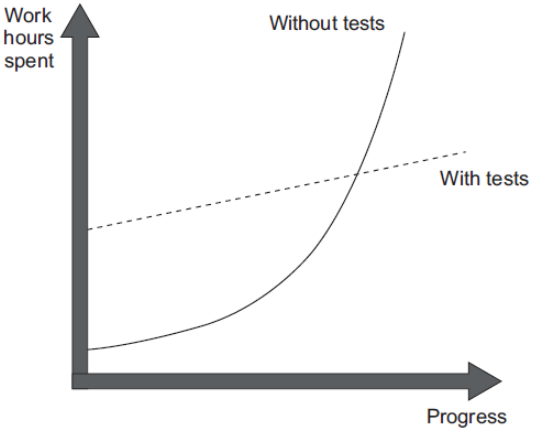
<Name>

There's much more to unit testing than the act of writing tests.

—***Khorikov*, Unit Testing Principles, Practices, and Patterns, 3**

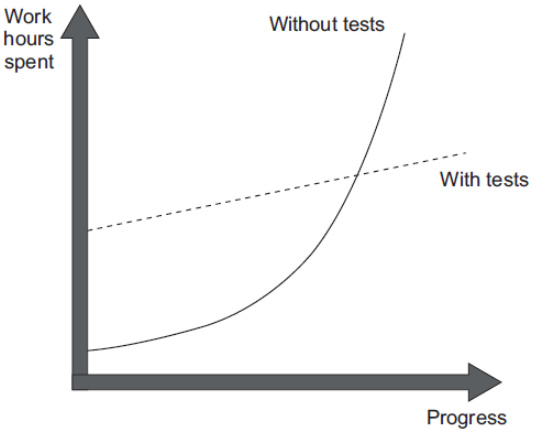
The goal of unit testing

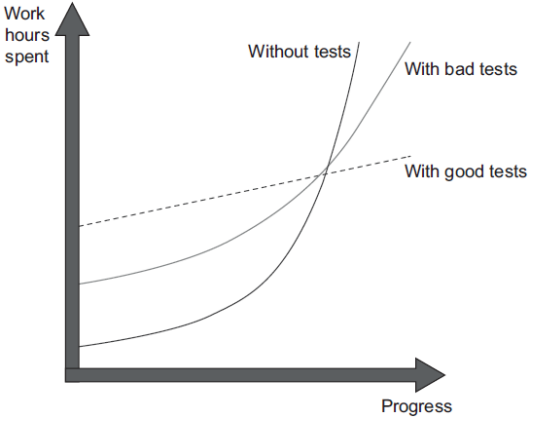
To enable **sustainable** growth of software project.



The goal of unit testing

To enable **sustainable** growth of software project.





Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

```
1  def is_fizzbuzz(num: int) -> bool:
2      if num % 3 and num % 5:
3          return True
4      return some_var
5
6  def test_fizzbuzz():
7      result = is_fizzbuzz(3)
8      assert result
```

$$\frac{\textit{Number of statements executed}}{\textit{Total number of statements}} \approx 67\%$$

Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

```
1  def is_fizzbuzz(num: int) -> bool:
2      return True if num % 3 and num % 5 else some_var
3
4  def test_fizzbuzz():
5      result = is_fizzbuzz(3)
6      assert result
```

$$\frac{\textit{Number of statements executed}}{\textit{Total number of statements}} = 100\%$$

Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

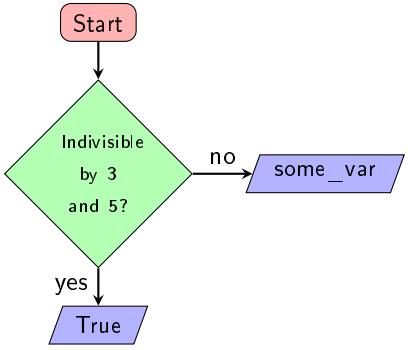
```
1  def is_fizzbuzz(num: int) -> bool:
2      return True if num % 3 and num % 5 else some_var
3
4  def test_fizzbuzz():
5      result = is_fizzbuzz(3)
6      assert result
```

$$\frac{\textit{Branches traversed}}{\textit{Total number of branches}} = 50\%$$

Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

```
1  def is_fizzbuzz(num: int) -> bool:
2      return True if num % 3 and num % 5 else some_var
3
4  def test_fizzbuzz():
5      result = is_fizzbuzz(3)
6      assert result
```

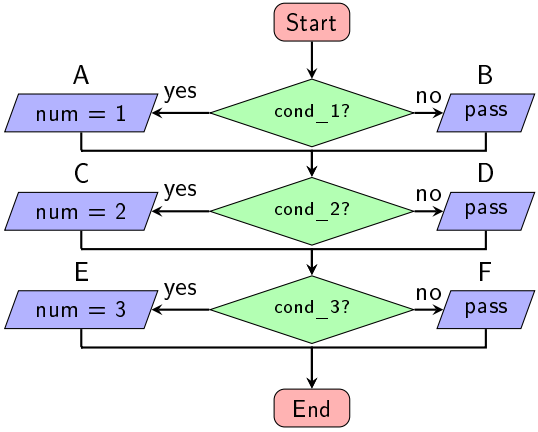


Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

```
1   def generate_number(
2       cond_1: bool = True,
3       cond_2: bool = True,
4       cond_3: bool = True,
5   ) -> int:
6       if cond_1:
7           num = 1
8       if cond_2:
9           num = 2
10      if cond_3:
11          num = 3
12      return num
```

Possible paths:
ACE, ACF, ADE, ADF, BCE, BCF, BDE, BDF



Coverage metrics

Statement vs Branch vs ~~Path~~ vs ~~Condition~~

```
1  def is_fizzbuzz(num: int) -> bool:
2      if num % 3 and num % 5:
3          return True
4      return some_var
5
6  def test_fizzbuzz():
7      result = is_fizzbuzz(3)
8      assert result
```

num % 3	num % 5	num % 3 and num % 5
True	True	True
True	False	False
False	True	False
False	False	False

[C]overage metrics are a good negative indicator, but a bad positive one.

—***Khorikov***, **Unit Testing Principles, Practices, and Patterns**, ***15***

Definition of a unit test

- Verifies a small piece of code,
- Does it quickly, and
- Does it in an isolated manner.

An integration test is a test that doesn't meet one of these criteria. End-to-end tests are a subset of integration tests and usually include more dependencies.

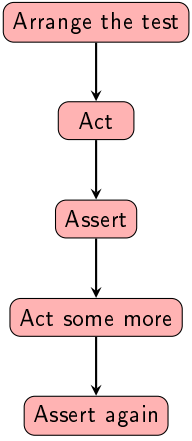
Anatomy of a unit test

The AAA (3A) pattern, also Given-When-Then pattern.

```
1  # A cohesive set of tests, optional
2  class TestCalculator:
3
4      # Name of the unit test
5      def test_sum_of_two_numbers(self):
6          # Arrange
7          first = 10
8          second = 20
9          calculator = Calculator()
10
11         # Act
12         result = calculator.sum(first, second)
13
14         # Assert
15         assert result == 30
```

- In *Arrange*, bring the system under test (SUT) to the a desired state
- In *Act*, call the method on the SUT, pass the prepared dependencies, and capture the output (if any).
- In *Assert*, verify the outcome. The outcome could be the return value, the final state of the SUT, or the methods the SUT called on its collaborators.

Things to avoid for unit tests



- Avoid multiple arrange, act, and assert sections.

Things to avoid for unit tests

```
def test_node_with_python_updates(self, req_file):
    with TestCase.assertLogs("...logger") as cap:
        assert check_requirements(
            NODE, req_file
        ) == 2
        for i, rec in enumerate(cap.records):
            idx = int(i / 2)
            if i == 4:
                assert (
                    ''2 packages updated''
                    in rec.getMessage()
                )
            elif i % 2 == 0:
                assert (
                    f''{PY_PKGS[idx]} not found''
                    in rec.getMessage()
                )
            else:
                assert (
                    f''pip install {PY_PKGS[idx]}''
                    in rec.getMessage()
                )
```

- Avoid multiple arrange, act, and assert sections.
- Avoid `if` statements.