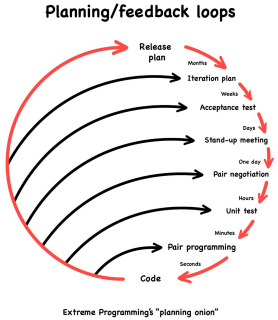
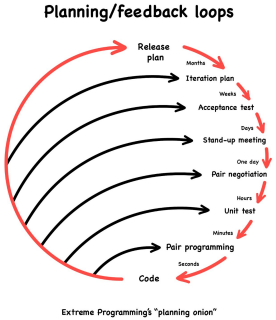


Test-Driven Development

Mobina Noori - Melika Bahrani

Prof : Dr. Hasheminejad

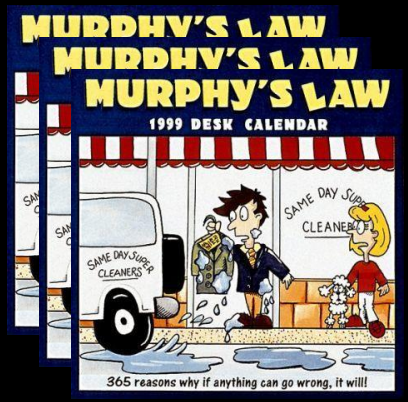


THE TWO STATES OF EVERY PROGRAMMER

I AM A GOD.

I HAVE NO IDEA WHAT I'M DOING.

Murphy's Law of Debugging: The thing you believe so deeply can't possibly be wrong so you never bother testing it is definitely where you'll find the bug after you pound your head on your desk and change it only because you've tried everything else you can possibly think of.



We know how skyscrapers work!

Just give us some bricks & we'll get started!!

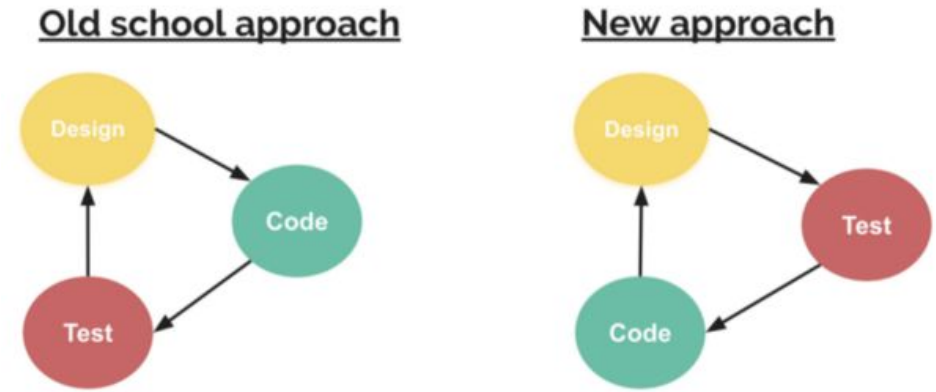
Software Engineering

vs

Programming

Table Of Contents

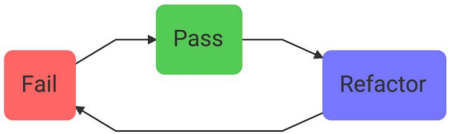
What is TDD?

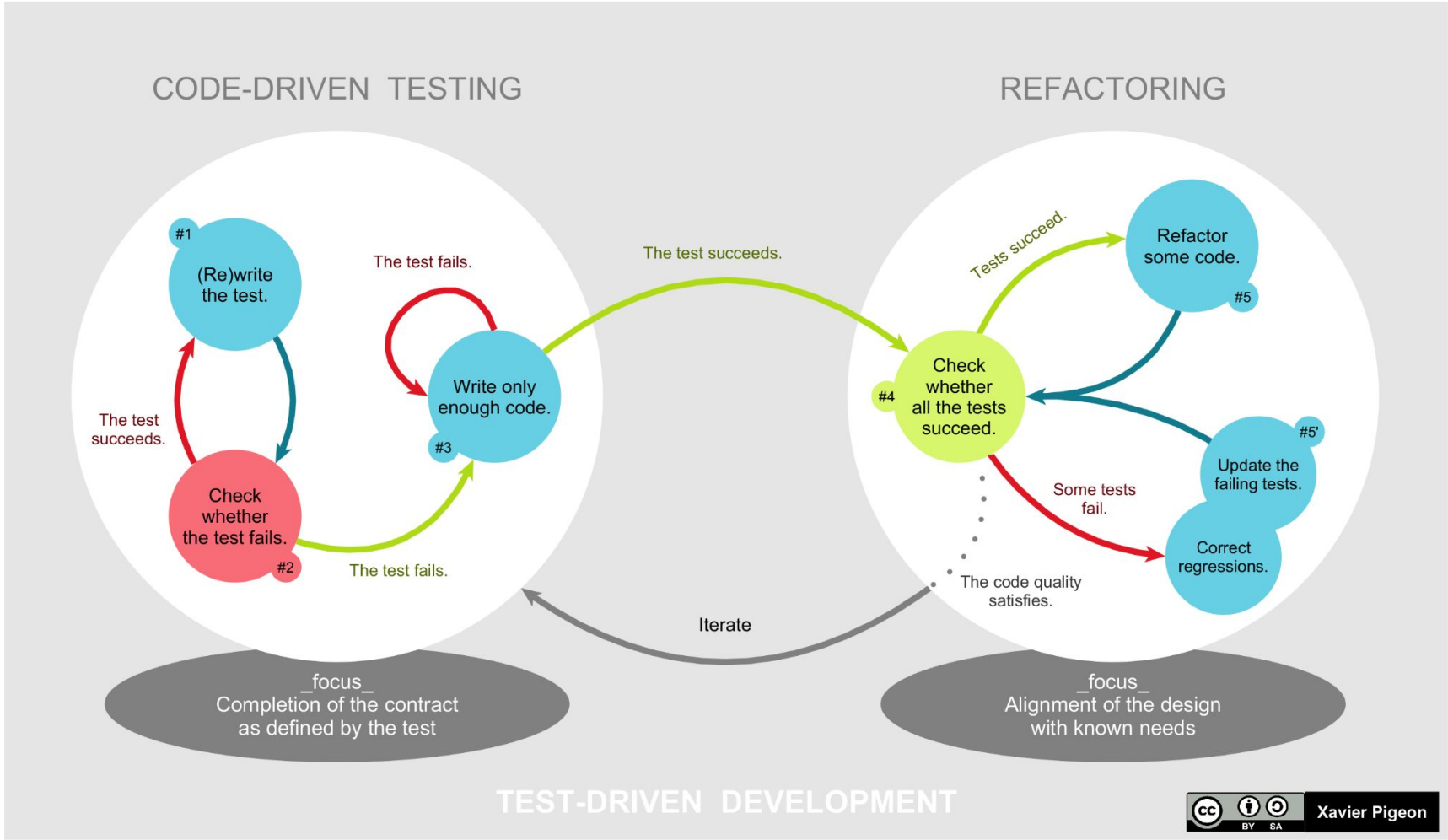

Old school approach
Design
Code
Test
New approach
Design
Test
Code

A. Before you write implementation code, write some code that proves that the implementation works or fails. Watch the test fail before moving to the next step (this is how we know that a passing test is not a false positive — how we test our tests).

B. Write the implementation code and watch the test pass.

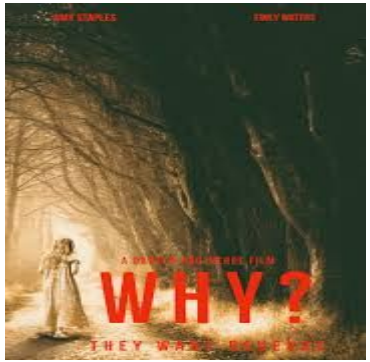
C. Refactor if needed. You should feel confident refactoring your code now that you have a test to tell you if you've broken something.





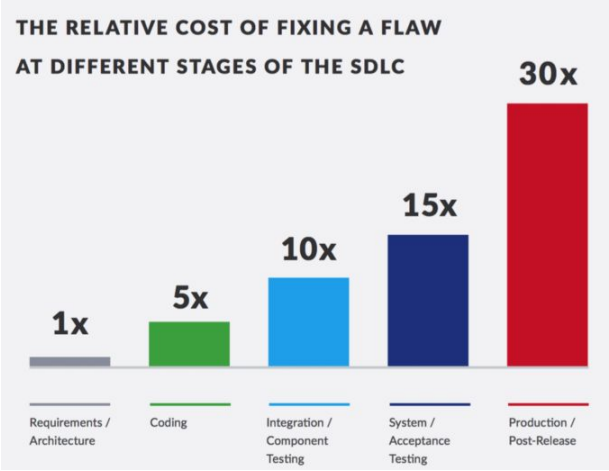
Why TDD?

....



- Better program design and higher code quality.
- TDD improve test coverage, which leads to 40%-80% fewer bugs in production. It's like a giant weight lifting off your shoulders.
- Bugs still happen, of course, but the releases show production bugs have dropped to near zero.
- TDD eradicates fear of change.

- Detailed project documentation.
- TDD reduces the time required for project development.
- Code flexibility and easier maintenance.
- Save project costs in the long run.

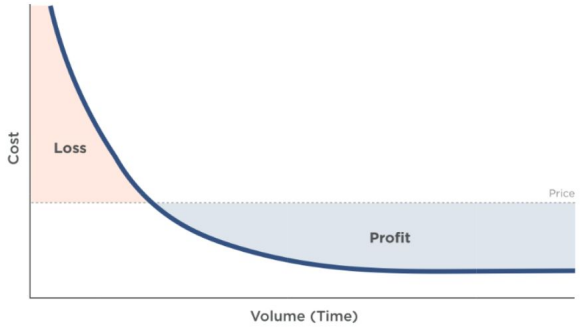


How TDD Can Save You Development Time?

- TDD has a learning curve, and while you're climbing that learning curve, it can and frequently does add 15% – 35% to implementation times. But at some point in the learning curve, something magical will begin to happen → coding faster with unit tests than you ever did without them.

- The point is not how long it takes to type this code. The point is how long it takes to debug it if something goes wrong.

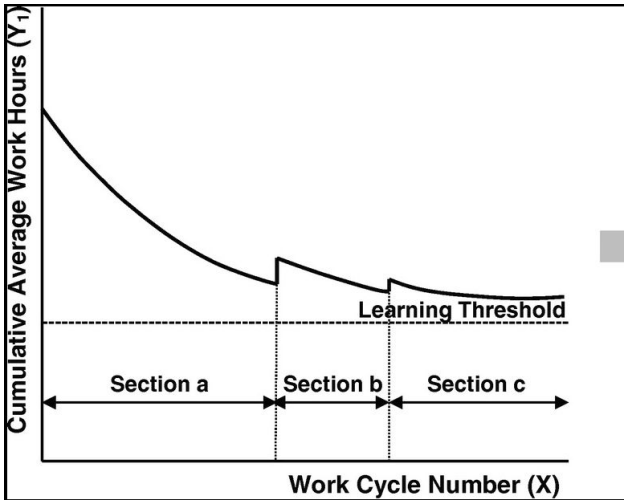
The Manufacturing Learning Curve (MLC)



The Big Idea

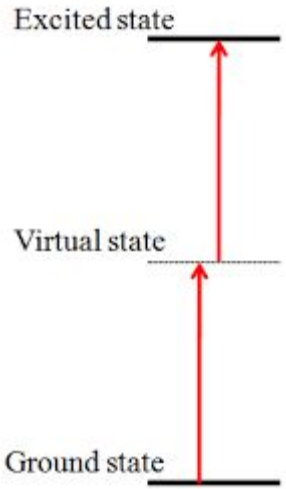
The Manufacturing Learning Curve tracks the cost of creating goods over time.

- Cost decreases as volume increases
- Shape of curve differs by industry
- Learning is non-predictive
- Cost accounting gives us the points to plot



TDD is far more than a safety net. It's also constant, fast, real time feedback. Instant gratification when I get it right. Instant, descriptive bug report when I get it wrong.

How to Write Better Code With \TDD/?



How to write much simpler code that is easier to extend, maintain, and scale, both in complexity, and across large distributed systems like cloud infrastructure.

How TDD Saves Whole Teams Time?

- That process builds a safety net that few bugs will slip through, and that safety net has a magical impact on the whole team. It eliminates fear of the merge button.

- Without fear of change, the development cadence runs a lot smoother. Pull requests stop backing up. Your CI/CD is running your tests — it will halt if your tests fail. It will fail loudly, and point out what went wrong when it does.

And that has made all the difference.

Frameworks for Test Driven Development

Based on unique programming languages, there are multiple frameworks that support test driven development. Listed below are a few popular ones.

- csUnit and NUnit – Both are open source unit testing frameworks for .NET projects.
- PyUnit and DocTest: Popular Unit testing framework for Python.
- Junit: Widely used unit testing tool for Java.
- TestNG: Another popular Java testing framework. This framework overcomes the limitations of Junit.
- Rspec: A testing framework for Ruby projects


Learn TDD In Practice!

1. Test Driven Development

2. Write Unittest For Written Code!

```
import unittest
from mycode import *

class MyFirstTests(unittest.TestCase):

def test_hello(self):
        self.assertEqual(hello_world(), 'hello world')
```

```
def hello_world():
    pass
```

```
F

======================================================================

FAIL: test_hello (__main__.MyFirstTests)

----------------------------------------------------------------------

Traceback (most recent call last):

File "mytests.py", line 7, in test_hello

self.assertEqual(hello_world(), 'hello world')

AssertionError: None != 'hello world'

----------------------------------------------------------------------

Ran 1 test in 0.000s

FAILED (failures=1)
```

```python
def hello_world():
    return 'hello world'
```

```
.

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK
```

```
import unittest
from mycode import *

class MyFirstTests(unittest.TestCase):

def test_hello(self):
        self.assertEqual(hello_world(), 'hello world')

    def test_custom_num_list(self):
        self.assertEqual(len(create_num_list(10)), 10)
```

```
def hello_world():
    return 'hello world'

def create_num_list(length):
    pass
```

```
E.

======================================================================

ERROR: test_custom_num_list (__main__.MyFirstTests)

----------------------------------------------------------------------

Traceback (most recent call last):

File "mytests.py", line 14, in test_custom_num_list

self.assertEqual(len(create_num_list(10)), 10)

TypeError: object of type 'NoneType' has no len()

----------------------------------------------------------------------

Ran 2 tests in 0.000s

FAILED (errors=1)
```

```python
def hello_world():
    return 'hello world'

def create_num_list(length):
    return [x for x in range(length)]
```

```
..

----------------------------------------------------------------------

Ran 2 tests in 0.000s

OK
```

```
import unittest
from mycode import *

class MyFirstTests(unittest.TestCase):

def test_hello(self):
        self.assertEqual(hello_world(), 'hello world')

def test_custom_num_list(self):
        self.assertEqual(len(create_num_list(10)), 10)

    def test_custom_func_x(self):
        self.assertEqual(custom_func_x(3,2,3), 54)
```

```
def hello_world():
    return 'hello world'

def create_num_list(length):
    return [x for x in range(length)]

def custom_func_x(x, const, power):
    pass
```

```
F..

======================================================================

FAIL: test_custom_func_x (__main__.MyFirstTests)

----------------------------------------------------------------------

Traceback (most recent call last):

File "mytests.py", line 17, in test_custom_func_x

self.assertEqual(custom_func_x(3,2,3), 54)

AssertionError: None != 54

----------------------------------------------------------------------

Ran 3 tests in 0.000s

FAILED (failures=1)
```

```python
def hello_world():
    return 'hello world'

def create_num_list(length):
    return [x for x in range(length)]

def custom_func_x(x, const, power):
    return const * (x) ** power
```

```
...

----------------------------------------------------------------------

Ran 3 tests in 0.000s

OK
```

What To Read For More Details

- Test-Driven Development by Example (Kent Beck)
- Test-Driven Development with Python (Harry J.W. Percival)
- Python Doc (Unittest)
- More Details In Testing
- Best Practices
- From Apprentice To Master
- Best Practices
- More Samples

Conclusion

Processes might feel tedious initially but they are necessary to transition from a 'programmer' to a software engineer. And they pay off in better productivity, cleaner codebase and better designed system.