

INTRODUCTION TO

TEST DRIVEN DEVELOPMENT

For development teams who wish to write high quality software...

...TDD is a development methodology comprised of a **simple sequence of repeated steps** - red, green, refactor...

...that help reduce risk, make changes easier
- ultimately making development faster.

TDD enables developers to have a high degree of confidence in the intended behavior of software components...

...thereby **reducing the time** needed to understand code and making changes easier and less risky.



```
graph TD; A[1. Failing Test] --> B[2. Make Test Pass]; B --> C[3. Refactor]; C --> A;
```

1. Failing Test

2. Make Test Pass

3. Refactor

- Smaller types
- Forces small steps
- Limited responsibility
- Writing tests forces you to see the world from the perspective of a maintainer
- Loose coupling
- Know when you break stuff (high test coverage)
- Refactorings can be made with confidence

- Test code is production code and must be maintained
- Has a learning curve
- “Perceived” development velocity may reduce

[SetUp]

public void **SetUp()**

{

 _putApi = MockRepository.GenerateStub<PutApi>();

 _postApi = MockRepository.GenerateStub<PostApi>();

 _typeUnderTest = new MyType(_putApi, _postApi);

}

[Test]

public void **UsesPostApiWhenNewPrice()** { }

[Test]

public void **ShouldUsePostApiWhenNewPrice()** { }

[Test]

public void **Save_NewPrice_UsesPostApi()**

{

//arrange

 _putApi.Stub(d => d.MethodName(Arg<string>.Is.Anything)).Return("foo");

//act

 _typeUnderTest.Save();

//assert

 _postApi.AssertWasCalled(a => a.Save(Arg<string>.Is.Anything));

}

- Mocking/stubbing framework
- Unit test framework
- Pair programming
- IOC Container
- Default parameters
- Null object pattern

“GOOS”

www.growing-object-oriented-software.com