

Jacob Dyrvig Sørensen
Kasper Rask Sørensen

28/9/2018

EASV Computer Science

GitHub: <https://github.com/dyller/TestDrivenDevelopment>

Indhold

Interface	3
First test and solution	3
Reflections on the test first approach and optimal pair-Programming	6

Interface

```
namespace TDDProject
{
    public interface IStackOfInt
    {
        //insert number in top of the stack
        void Push(double x);
        //delete first number in stack
        double Pop();
        //return first number in stack
        double Top ();
        // clear all numbers in stack
        void Clear();
        //Count number in stack
        int Count ();
        //Adds the 2 first number in the stack and put in the top of the stack
        void AddTop2();
        //subtract the 2 first number in the stack and put in the top of the stack
        void SubTop2();
        //Multiply the first 2 number in the stack and put in the top of the stack
        void Multiplytop2();
        //Devide the first 2 number in the stack and put in the top of the stack
        void DivisionTop2();
        //Add all number in the stack and put it to the stack
        void AddAll();
        //Multiply all number in the stack and put it to the stack
        void MultiAllNumbers();
    }
}
```

This is our interface for all the method their should be implemented in the class.

First test and solution

Push(double value).




The way push should work, is that it should take the value and insert at top of the stack.
So the first step we took, was to check after we pushed a number, that it is at the top of the stack.

```

[TestMethod]
public void TestPushCheckInTop()
{
    StackOfInt SOI = new StackOfInt();
    double expResult = 2;
    SOI.Push(expResult);
    double result = SOI.StackInt.Peek();
    Assert.AreEqual(expResult, result);

    double expResult2 = 3;
    SOI.Push(expResult2);
    double result2 = SOI.StackInt.Peek();
    Assert.AreEqual(expResult2, result2);
}

```

 TDDProject (1 tests) 1 failed
 UnitTestStackOfInt (1) 353 ms
 UnitTestStackOfInt (1) 353 ms

The reason we did it two times, was to be sure, that it wasn't only working with 1 number because that already first in the stack.

```

public void Push(double x)
{
    StackInt.Push(x);
}

```

This is our solution to first test, and it passed the test after, so we decided not to make more test to this method, we were thinking it would be to most to test if the Stack was still saved like before just with the number we pushed at the top.

Pop.

Pop is a method that should remove the first number from the stack, and return the number it removes.

```

[TestMethod]
public void Testpop()
{
    StackOfInt SOI = new StackOfInt();

    double expResult=2 ;
    SOI.Push(expResult);
    SOI.Push(3);
    double expResult3 = SOI.Pop();
    double result = SOI.StackInt.Peek();
    Assert.AreEqual(expResult, result);
    Assert.AreEqual(expResult3, 3);

    double expResult2 = 3;
    SOI.Push(expResult2);
    SOI.Push(2);
    double expResult4= SOI.Pop();
    double result2 = SOI.StackInt.Peek();
    Assert.AreEqual(expResult2, result2);
    Assert.AreEqual(expResult4, 2);
}

```

So in this case this need to be tested for 2 thing, for returning the value and that it removes the first number.

```

public double Pop()
{
    return StackInt.Pop();
}

```

This was the solution for pop, and that passed the test.

Test Exception

<pre> [TestMethod] public void TestMultiplyTop2Exception() { StackOfInt SOI = new StackOfInt(); SOI.Push(1); try { SOI.Multiplytop2(); Assert.Fail(); } catch (Exception) { } } </pre>	<pre> public void SubTop2() { if (StackInt.Count > 1) { double result = StackInt.Pop() - StackInt.Pop(); StackInt.Push(result); } else { throw new Exception(); } } </pre>
---	---

The picture to the left is the test, in this test we will test if it gives an exception if we try to subtract 2 number, but only have one in the stack. So we use try catch, so it will catch if we get this exception, and if we are not getting that, it will fail the test.

On the right you can see the solution so here we check if there is 2 or more in the stack, and if that's true it will work, and if it's not true it throws an exception, so now the test will work.

We made this test for multiply2, add2, subtract2 and divide2.

Reflections on the test first approach and optimal pair-Programming

We started out by coding the interface at first, and then went on to doing the test individually, every time checking that the test could run, by having them fail, and then programming a method, so that the tests would pass.

Pros about TDD would absolutely be that it minimizes room for error, it helps you catch a small error before it becomes a larger issue later on. Cons might be that it takes a bit longer, and that you might not entirely know how to do the test at first, as your method could change over time.

Pros about pair programming is that it allows us to focus our resources on one problem, which also helps eliminate small mistakes like typos or using the wrong method in places where it shouldn't be.

However, doing pair programming also slows you down, because if you had 2 guys doing their own thing on different computers, this might speed up the process quite a lot. But as we're not that experienced yet, pair programming works quite well for us.