TDD

Crawling – Babysteps - Walking

Premise and initial thoughts

Looking forward to learn TDD

Mob programming

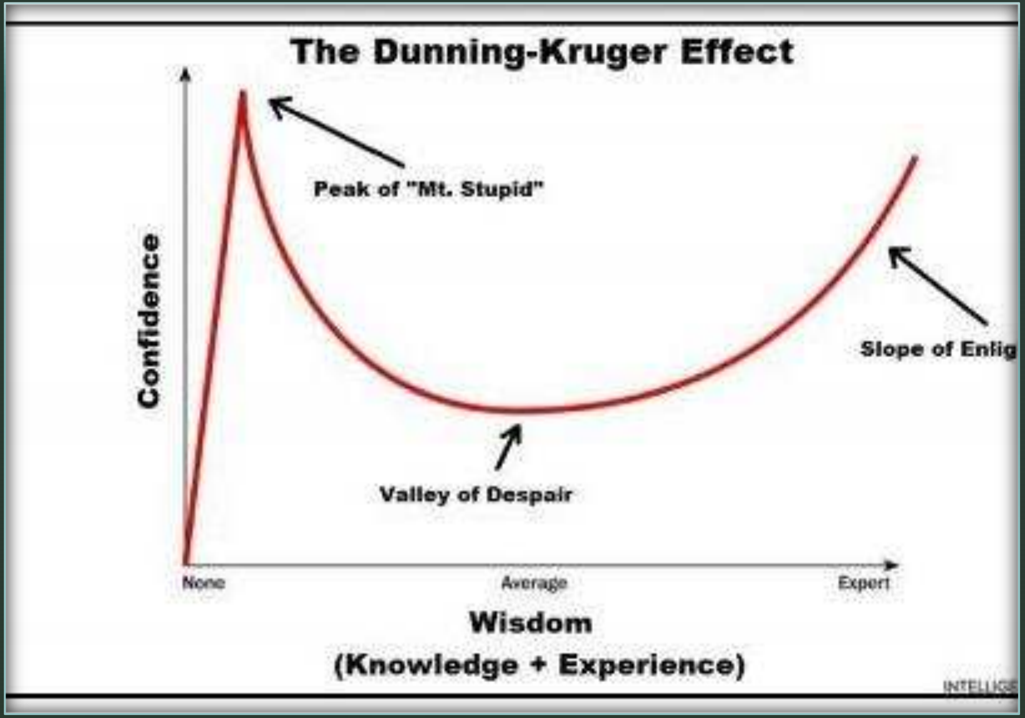
Presentation day

So what happened?

We made the first stumbling steps, we made the first failing test

Everyone enjoys a good mountain hike...sometimes.

Found our rhythm, our flow.



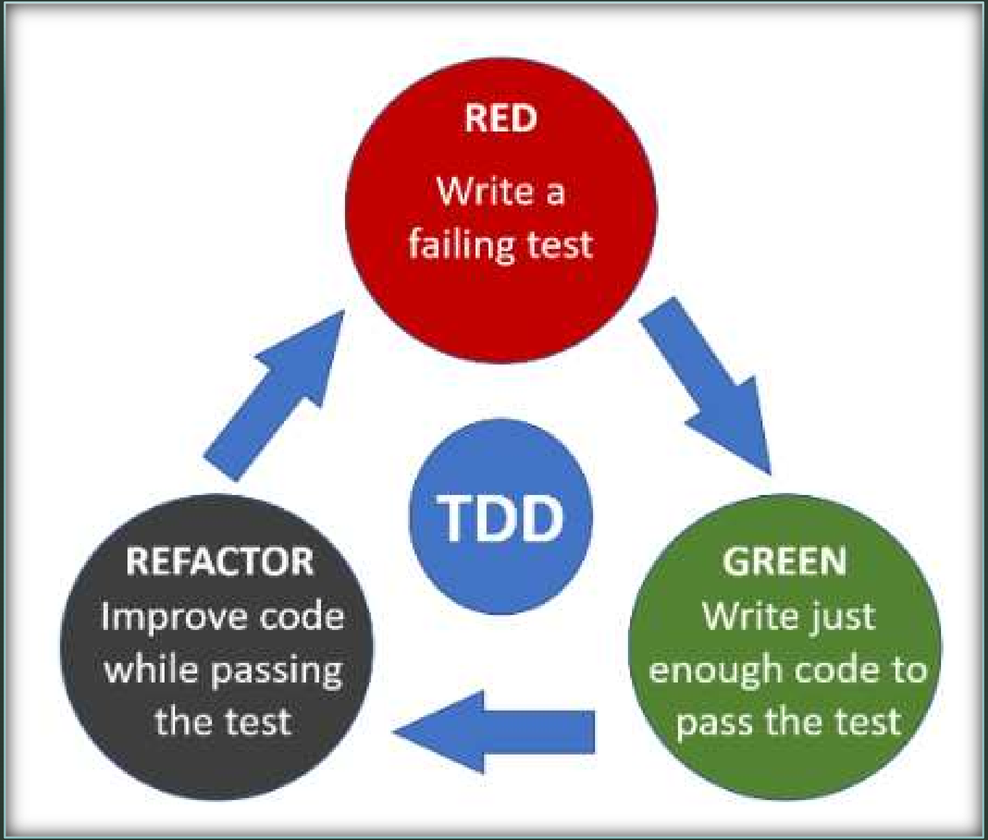
Ok, so what did we learn?

- Baby steps (3 laws of TDD)
- Red – Green – Refactor

  Fail it – Nail it – Work it
- Why do it the hard way, when you can be lazy?

  "Stop, stop right there, sooo, if you press ....."

  Quote: Alessandro x n!

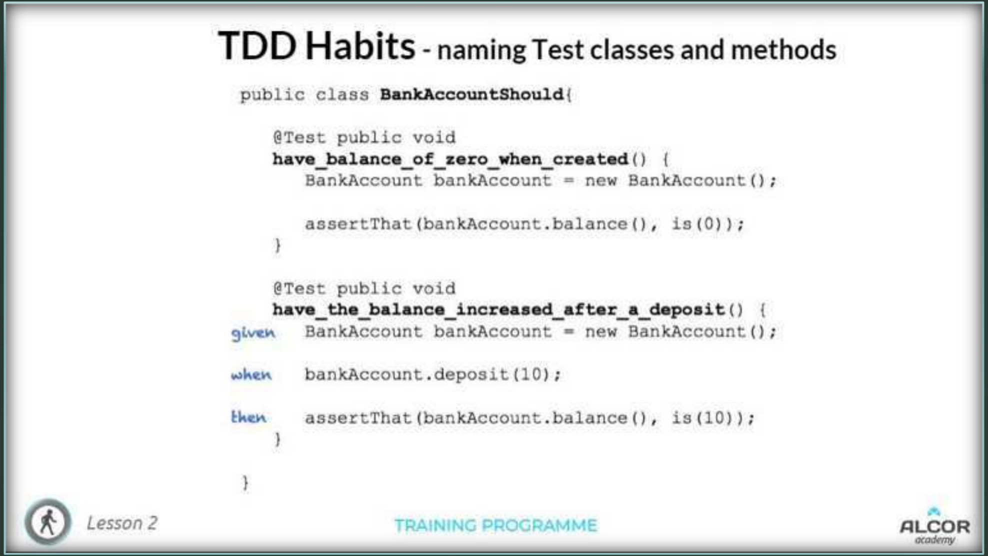

RED
Write a failing test
GREEN
Write just enough code to pass the test
REFACTOR
Improve code while passing the test
TDD

FIRST principals

- Fast
  The tests will be run very often.
- Isolated
  There should be no dependency between tests. Could be run in any order at any time.
- Repeatable
  They should always have the same result when run multiple times.
- Self validating
  Only two states: red or green. Absolutely no manual or human interpretation.
- Timely
  Must be written at the right time. BEFORE the code they're supposed to test.

How to name your tests

- SomethingShould
  - DoSomething
    - Given
      - When
        - Then



TDD Habits - naming Test classes and methods

```
public class BankAccountShould{

    @Test public void
    have_balance_of_zero_when_created() {
        BankAccount bankAccount = new BankAccount();

        assertThat(bankAccount.balance(), is(0));
    }

    @Test public void
    have_the_balance_increased_after_a_deposit() {
given   BankAccount bankAccount = new BankAccount();

when    bankAccount.deposit(10);

then    assertThat(bankAccount.balance(), is(10));
    }

}
```

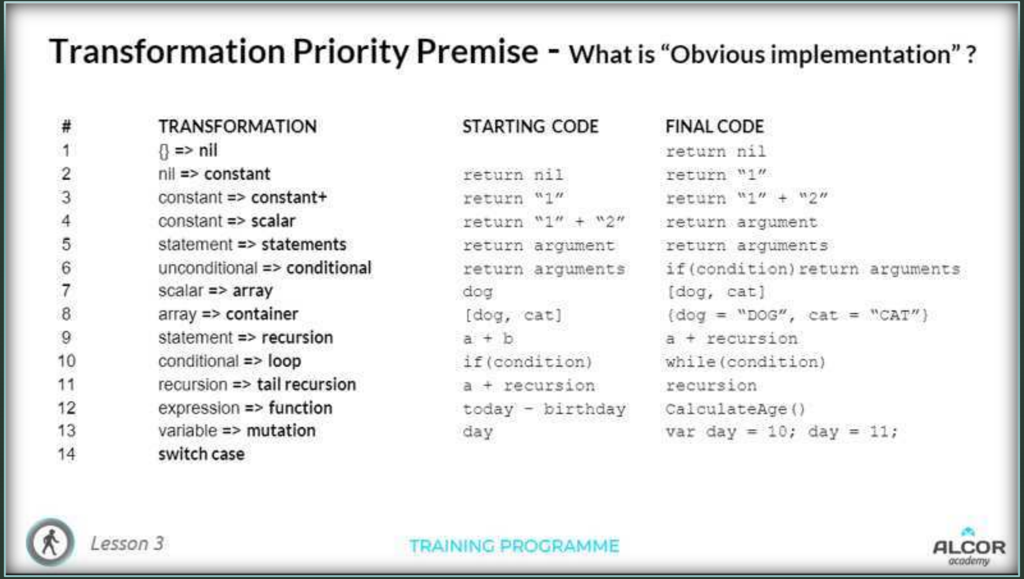
Lesson 2 TRAINING PROGRAMME ALCOR academy

Test smells

- Not testing anything
- Excessive setup
- Too many assertions
- Test to long
- Checking internals
- Checking more than strictly necessary
- Working only on dev machine
- Testing or containing irrelevant information
- Exception swallowing in test
- Test not belonging logically to the fixture
- Obsolete test
- Hidden functionality buried in the setup
- Bloated construction impeding test readability
- Unclear failing reason
- Conditional test logic

TPP – Transformation Priority Premise

- 1) Fake implementation
- 2) Obvious (simple) implementation
- 3) Triangulation with the next test



Transformation Priority Premise - What is “Obvious implementation” ?

#	TRANSFORMATION	STARTING CODE	FINAL CODE
1	{} => **nil**		`return nil`
2	nil => **constant**	`return nil`	`return “1”`
3	constant => **constant+**	`return “1”`	`return “1” + “2”`
4	constant => **scalar**	`return “1” + “2”`	`return argument`
5	statement => **statements**	`return argument`	`return arguments`
6	unconditional => **conditional**	`return arguments`	`if(condition)return arguments`
7	scalar => **array**	`dog`	`[dog, cat]`
8	array => **container**	`[dog, cat]`	`{dog = “DOG”, cat = “CAT”}`
9	statement => **recursion**	`a + b`	`a + recursion`
10	conditional => **loop**	`if(condition)`	`while(condition)`
11	recursion => **tail recursion**	`a + recursion`	`recursion`
12	expression => **function**	`today - birthday`	`CalculateAge()`
13	variable => **mutation**	`day`	`var day = 10; day = 11;`
14	**switch case**		

Lesson 3 TRAINING PROGRAMME ALCOR academy

Object Calisthenics rules

- 7/9 - Encapsulation
- Polymorphism
  Don't use else, and minimize use of conditional logic
- Follow naming standards
  No abbreviations

- Only one level of indentation per method
- Don't use the ELSE keyword
- Wrap all primitives and strings
- First class collections
  (wrap all collections)
- Only one dot per line
- No abbreviations
- Keep all entities small
- No classes with more than two instance variables
- No public getters/setters/properties

Thank you

For the "aha" moments

For learning me how to walk in the TDD world

For the mob programming experience

Ronny
Navelsaker
ronny.navelsaker
@bouvet.no

Sooo, when do we start running?