

# TDD - Test Driven Development

# A (re)introduction

---

STEFANO MONDINI – EOC

WALKING



# Introduction<sup>[1]</sup>

---

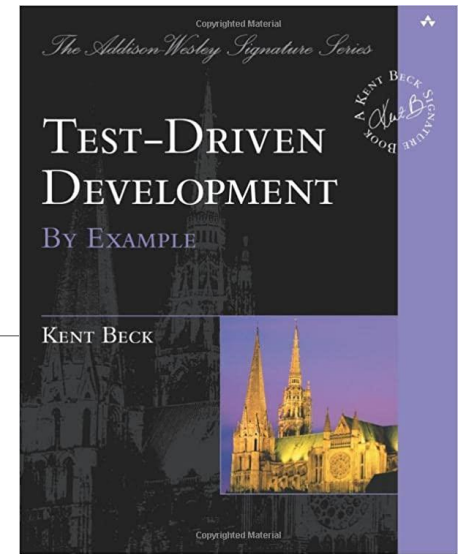


Kent Beck (1961)<sup>[2]</sup>

Original signer (1 of 17) of Agile Manifesto<sup>[3]</sup>

Author of the Extreme Programming books

Rediscoverer of Test-Driven Development



# Introduction

---

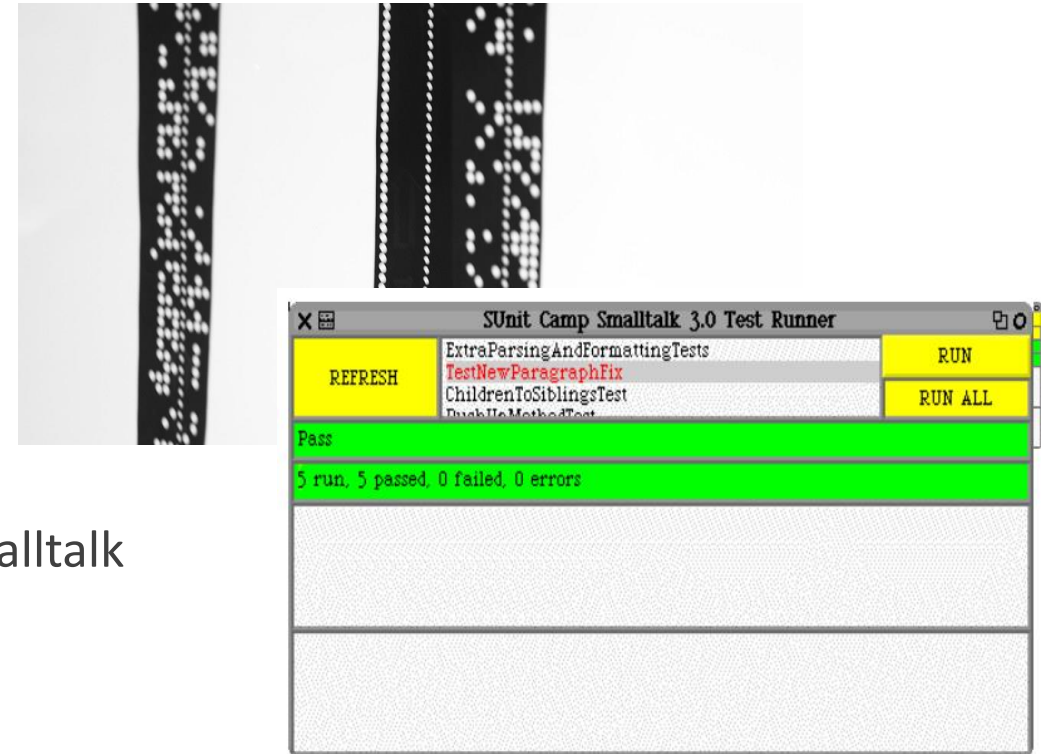
## 1970

Kent Beck: Read book about how to write programs

## 1990

Software consultant

Develop first framework for unit testing: SUnit for Smalltalk



# Write the test before I had the code

---



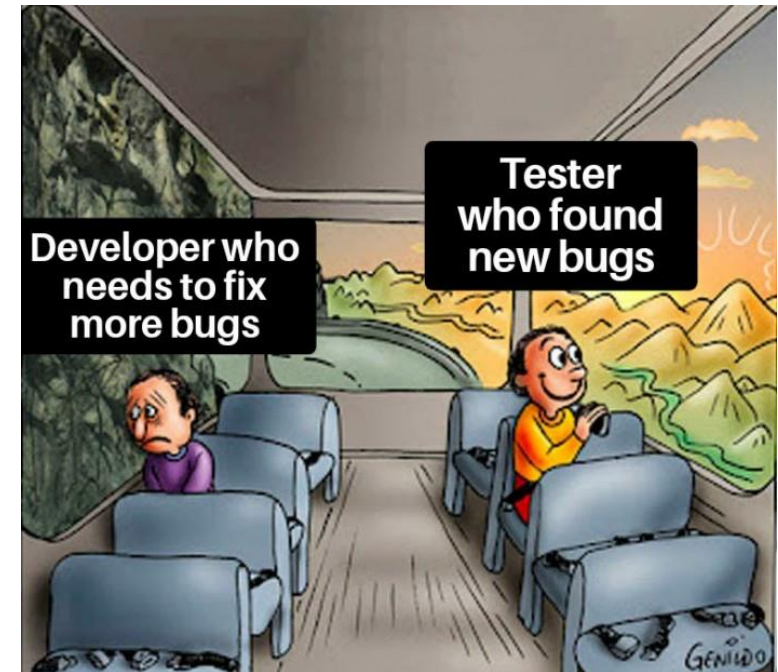
# Result

---

**Anxiety dropped away**

**I don't have to make all tests pass at once**

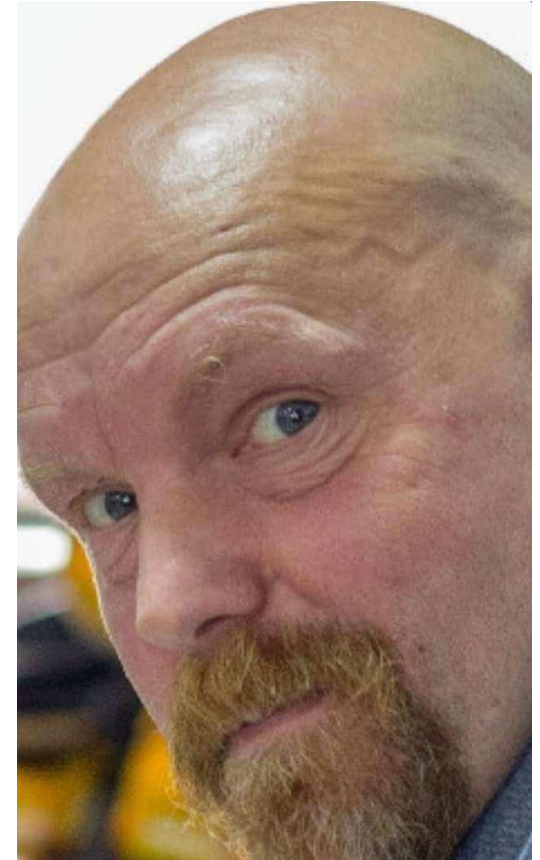
**Can I think of any other tests? No. I must be done**



# Two rules of TDD<sup>[4]</sup> – Kent Beck

---

- 1. Write new code only if an automated test has failed**
- 2. Eliminate duplication**



# TDD-Cycle

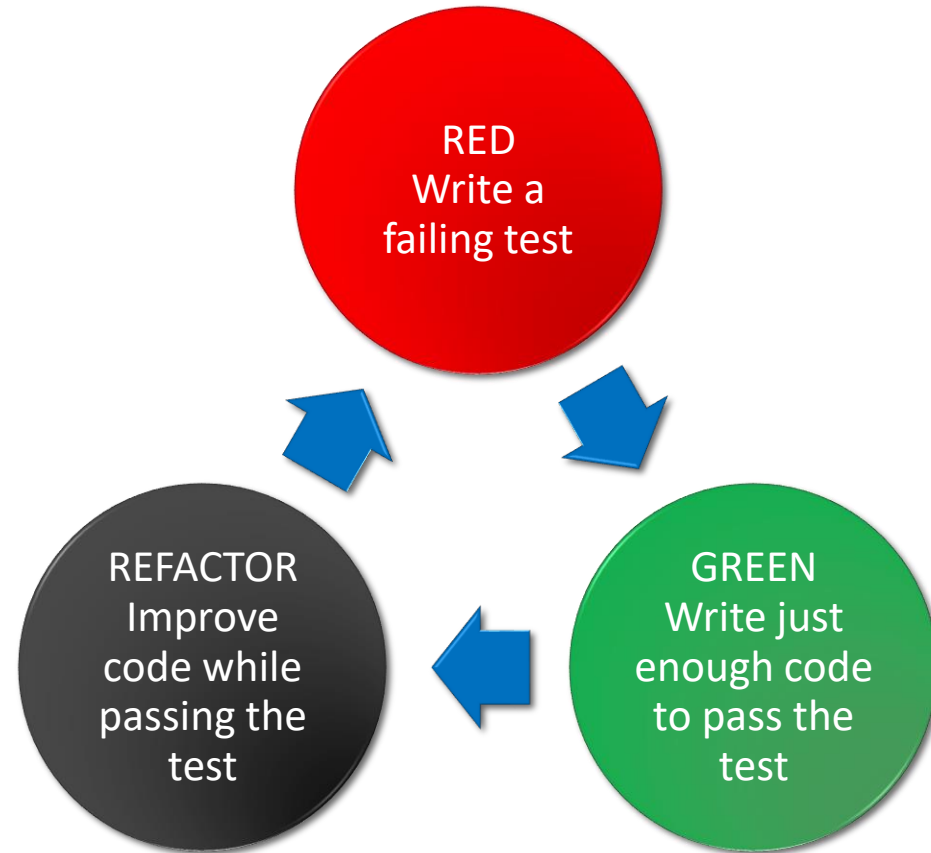
---

**Rhythm of code development**

**Focus on a **small scope** of our problem**

# TDD-Cycle - Red, Green, Refactor cycle

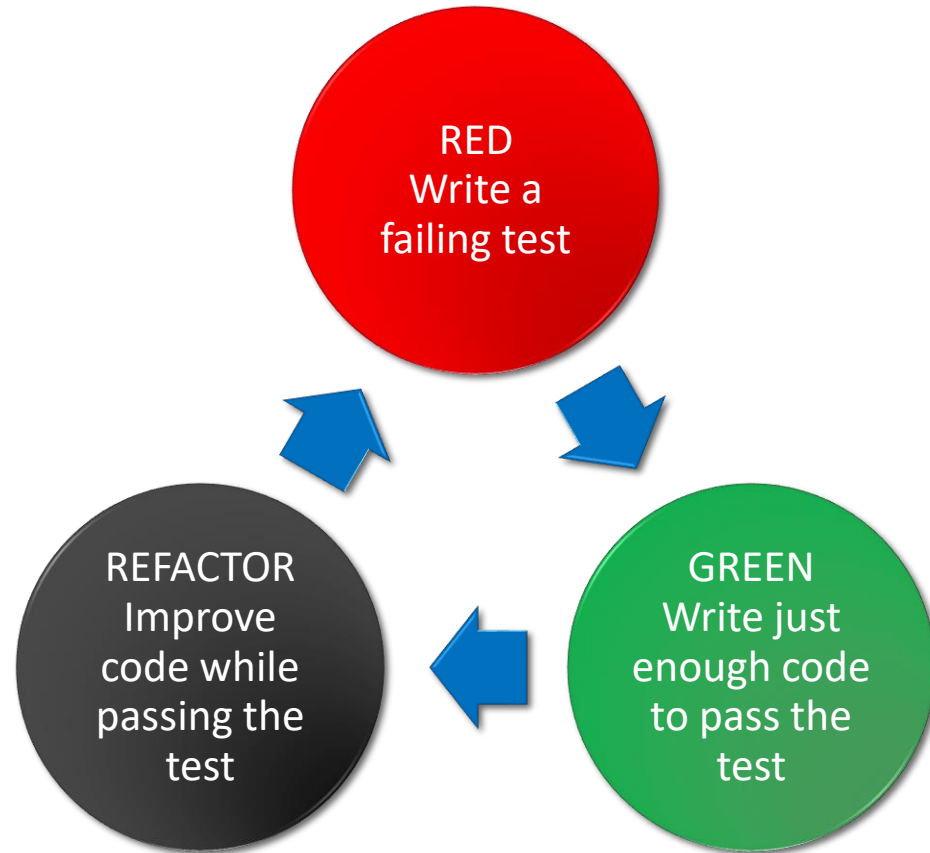
---





# TDD-Cycle - Red, Green, Refactor cycle

---



***Write a test***

***Make it run***

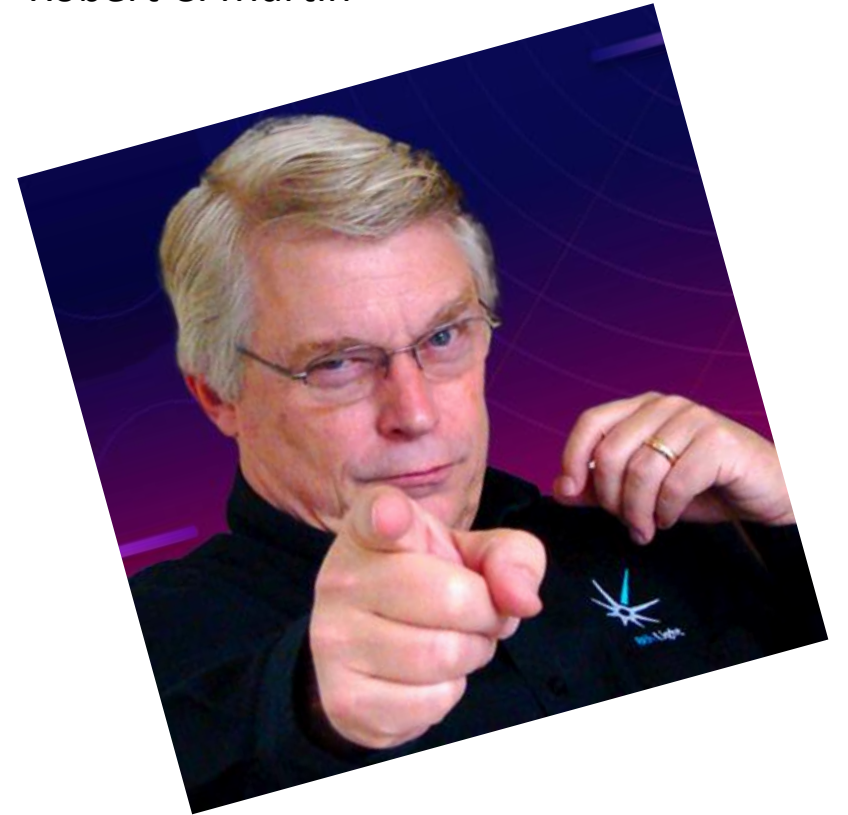
***Make it right***

# The three laws of TDD<sup>[5]</sup> – Uncle Bob

---

Robert C. Martin

1. No production code, **unless** make failing test pass
2. Only one test at time
3. No more code that is **sufficient** to pass the failing test



# Three steps for TDD<sup>[6]</sup> – Martin Fowler

---

1. Write a test
2. Write until test passes
3. Refactor **both new and old code** to make it well structured



# The three laws of TDD

---



RED  
Write a  
failing test

Write

Predict outcome

Run test

**See it fail**

# The three laws of TDD

---



Pass test **ASAP**

Commit code

Use hard-coded values

If statements

Fake it until you make it.

# Three techniques to make it “green”

---



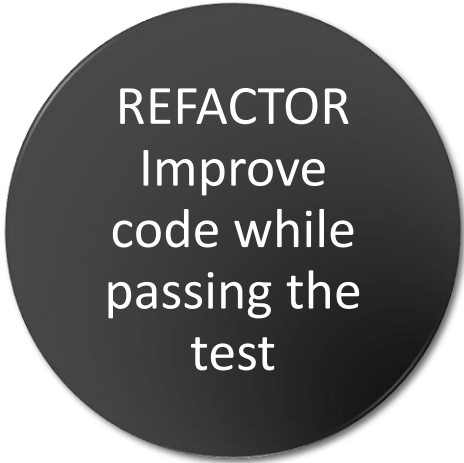
**Fake It**

**Obvious Implementation**

**Triangulation**

# The three laws of TDD

---



REFACTOR  
Improve  
code while  
passing the  
test

Improve design

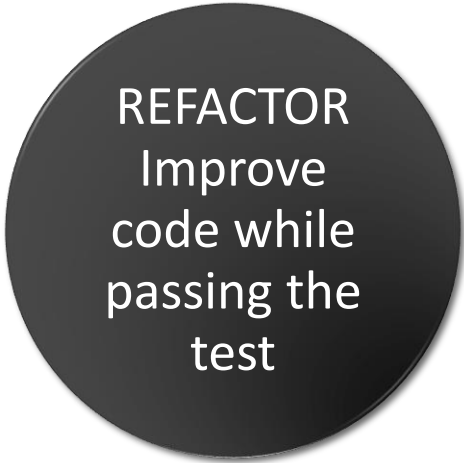
No change in behavior

Remove duplication

Improve names

# Refactoring: Rule of three

---

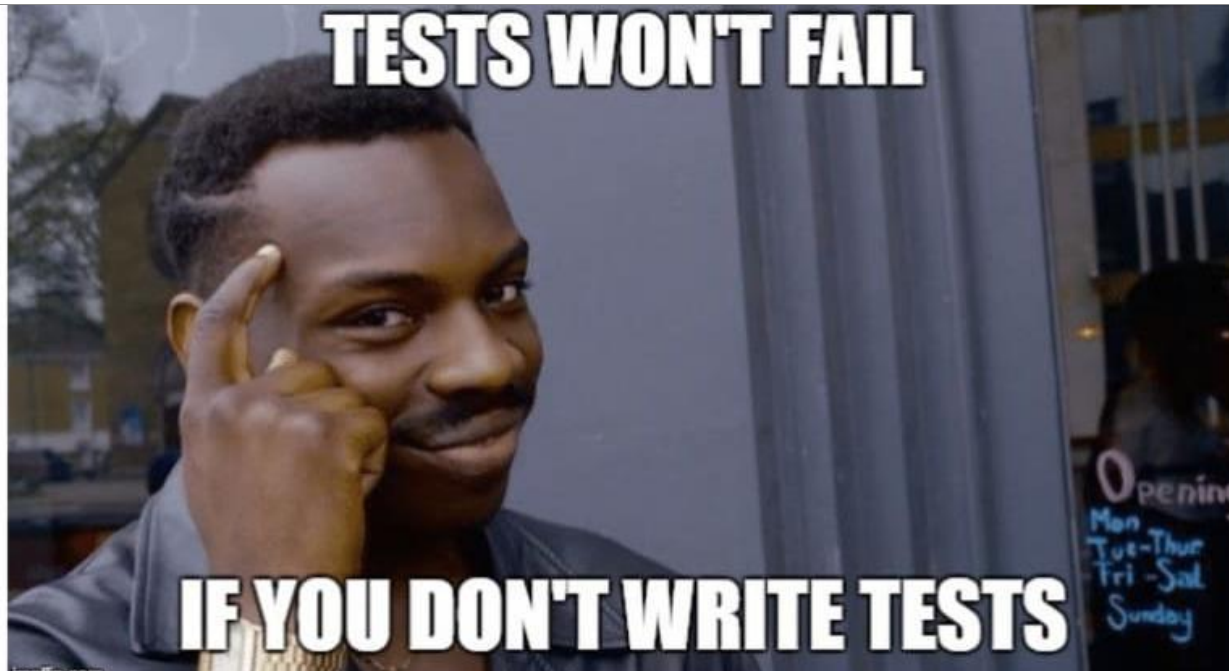


REFACTOR  
Improve  
code while  
passing the  
test

Rule of Three

Reduces risk of wrong  
abstraction





"Clean code that works (Ron Jeffries)"  
is the goal of Test-Driven Development

```
private bool IsEven(int number)
{
    //Added to pass unit test
    if (number == 11)
    {
        return false;
    }

    //Added to pass unit test
    if (number == 3)
    {
        return false;
    }

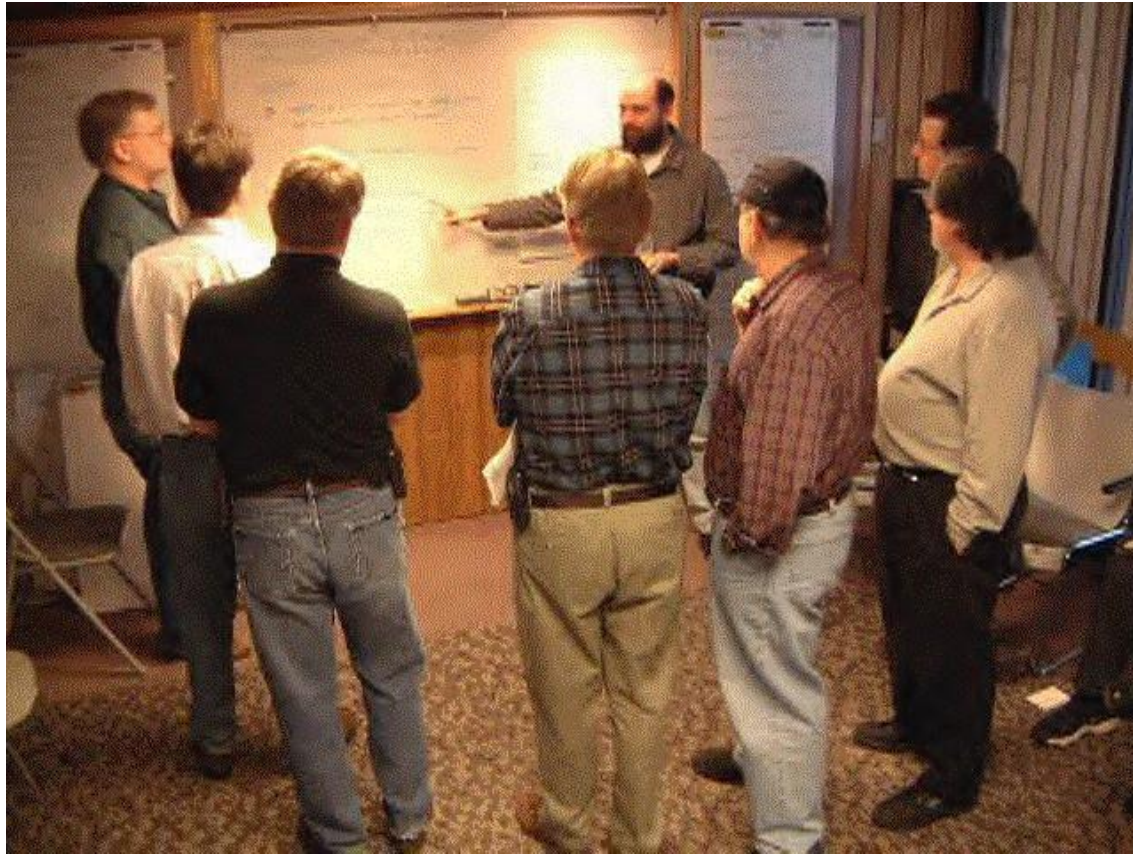
    //Fix for Ticket 12846
    if (number == 11407)
    {
        return false;
    }

    //Fix for Ticket 14336
    if (number == 9)
    {
        return false;
    }

    return true;
}
```

# Questions?

---



# Further reading and references

---

1. <https://tdd.mooc.fi/1-tdd>
2. <https://www.amazon.com/Test-Driven-Development-Kent-Beck/dp/0321146530>
3. <https://agilemanifesto.org/principles.html>
4. <https://stanislaw.github.io/2016/01/25/notes-on-test-driven-development-by-example-by-kent-beck.html>
5. <http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd>
6. <https://www.thedroidsonroids.com/blog/key-laws-of-tdd>

Immagini:

- ❖ <http://stephane.ducasse.free.fr/Programmez/OnTheWeb/Eng-Art8-SUnit-V1.pdf>
- ❖ Reddit