

Real Developers Don't Need Unit Tests

Mythbusting TDD myths



John Ferguson Smart
Wakaleo Consulting Ltd.

<http://www.wakaleo.com>

Email: john.smart@wakaleo.com

Twitter: [wakaleo](https://twitter.com/wakaleo)

Introduction

▶ Real Developers Don't Need Unit Tests



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Chuck Norris's code doesn't have bugs



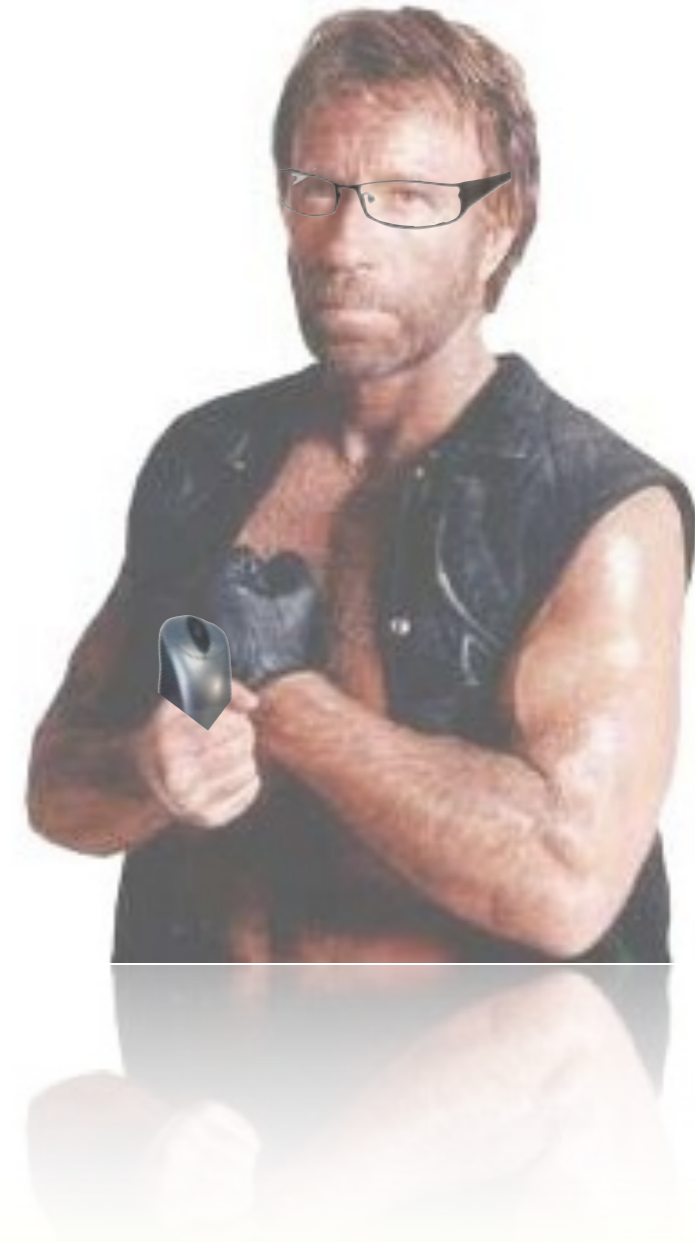
Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Chuck Norris's code doesn't have bugs

His code always work. ALWAYS.



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Chuck Norris's code is perfectly designed the first time round



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Chuck Norris's code is perfectly designed the first time round

His favorite design pattern is the Roundhouse Kick



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?



Introduction

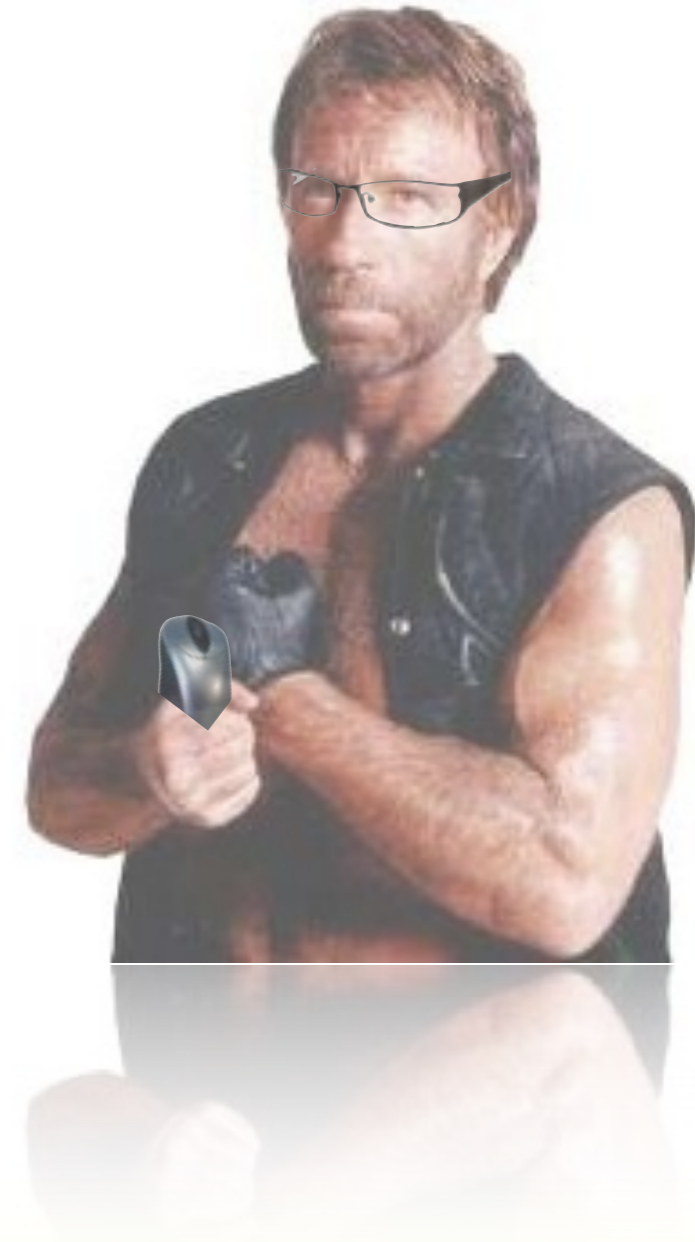
▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Chuck Norris doesn't need technical documentation



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Chuck Norris doesn't need technical documentation

He just stares down the code until it tells him everything he wants to know



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?

Chuck Norris doesn't need requirements to be executable



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?

Chuck Norris doesn't need requirements to be executable

He can execute whatever he wants



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?

Acceptance Tests?



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

Technical documentation?

Executable requirements?

Acceptance Tests?

Chuck Norris doesn't need
acceptance tests



Introduction

▶ Real Developers Don't Need Unit Tests

Bugs?

Emergent design?

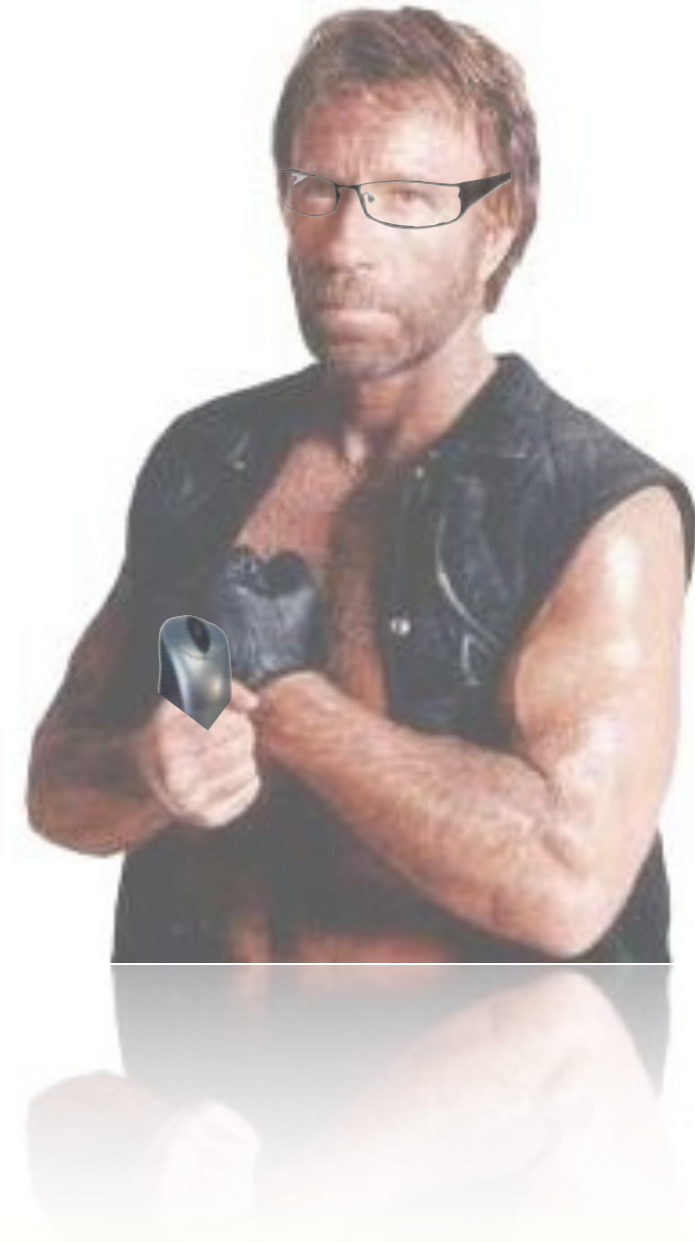
Technical documentation?

Executable requirements?

Acceptance Tests?

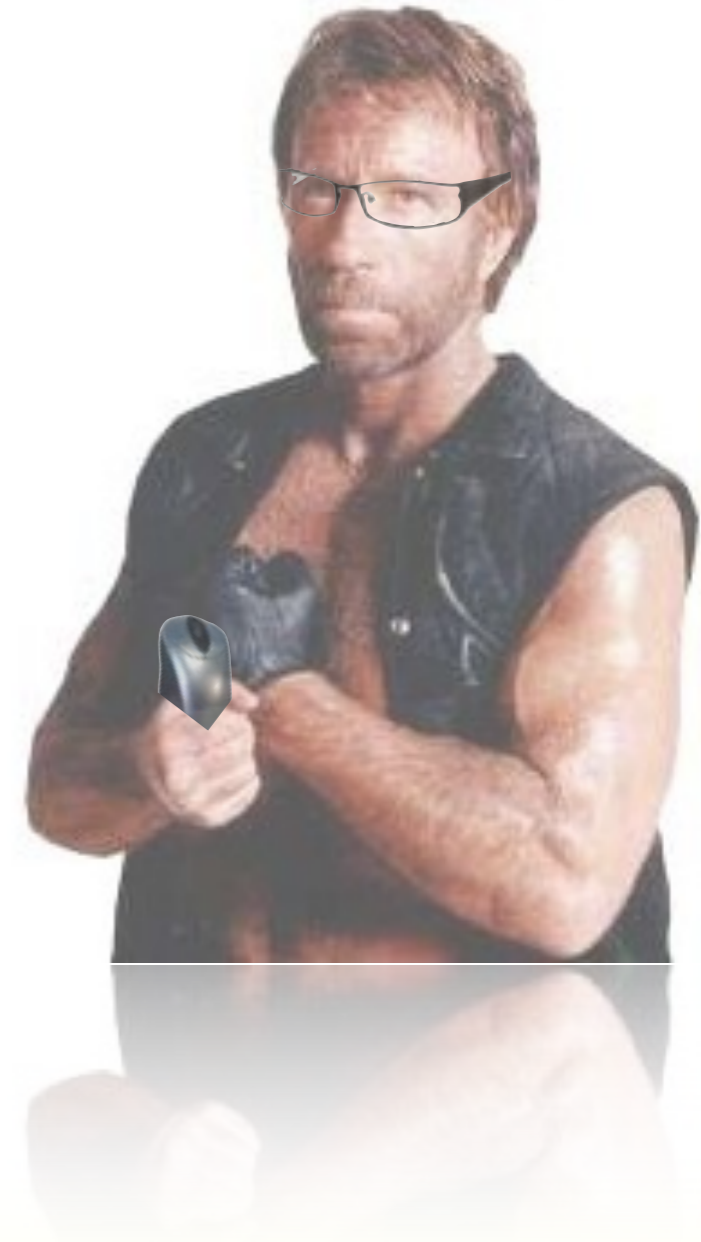
Chuck Norris doesn't need acceptance tests

No one refuses to accept Chuck



Introduction

▶ Real Developers Don't Need Unit Tests



Introduction

▶ Real Developers Don't Need Unit Tests

OK, so Chuck Norris doesn't need Unit Tests



Introduction

▶ Real Developers Don't Need Unit Tests

OK, so Chuck Norris doesn't need Unit Tests

But what about the rest of us?



TDD Basics



TDD Basics



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code
 - ▶ Have more confidence in our code



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code
 - ▶ Have more confidence in our code
 - ▶ Make changes more easily



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code
 - ▶ Have more confidence in our code
 - ▶ Make changes more easily
 - ▶ Write code that meets user requirements more accurately



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code
 - ▶ Have more confidence in our code
 - ▶ Make changes more easily
 - ▶ Write code that meets user requirements more accurately
(and incidentally...)



So what is this TDD thing, anyway?

TDD Basics

- ▶ TDD is *not* about writing tests
- ▶ TDD is a design strategy:
 - ▶ Write better-designed code
 - ▶ Have more confidence in our code
 - ▶ Make changes more easily
 - ▶ Write code that meets user requirements more accurately
(and incidentally...)
 - ▶ Build up a comprehensive set of automated tests



So what is this TDD thing, anyway?

TDD Basics

▶ The TDD development process

*“Never write a single line of code unless you have a failing automated test.
Eliminate duplication.”*

- Kent Beck

TDD Basics

▶ The TDD development process

*“Never write a single line of code unless you have a failing automated test.
Eliminate duplication.”*

- Kent Beck

TEST

Write a failing unit test. Ensure that it fails.

TDD Basics

▶ The TDD development process

*“Never write a single line of code unless you have a failing automated test.
Eliminate duplication.”*

- Kent Beck

TEST

Write a failing unit test. Ensure that it fails.

CODE

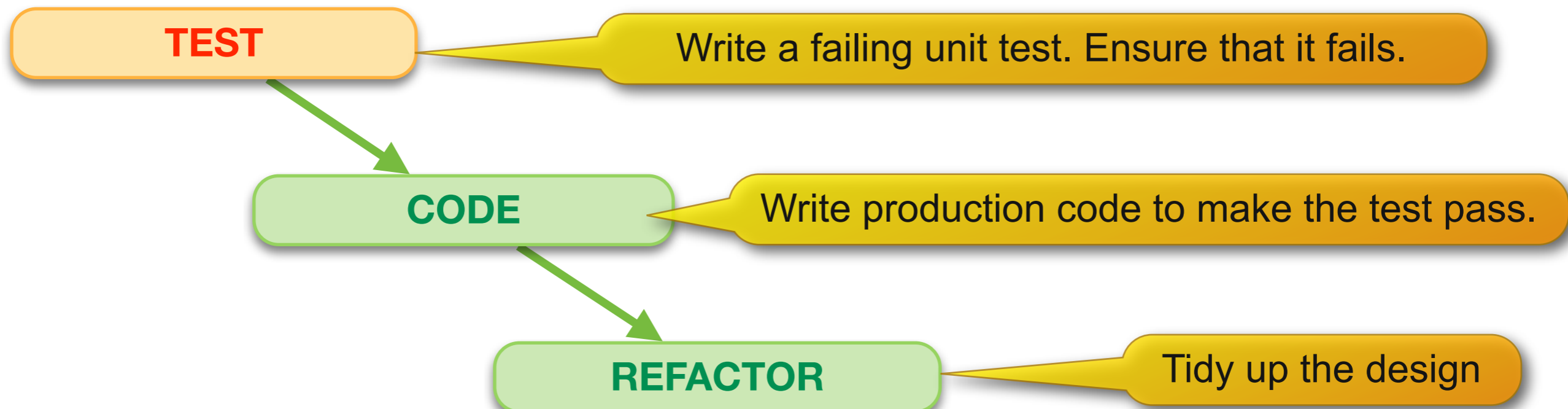
Write production code to make the test pass.

TDD Basics

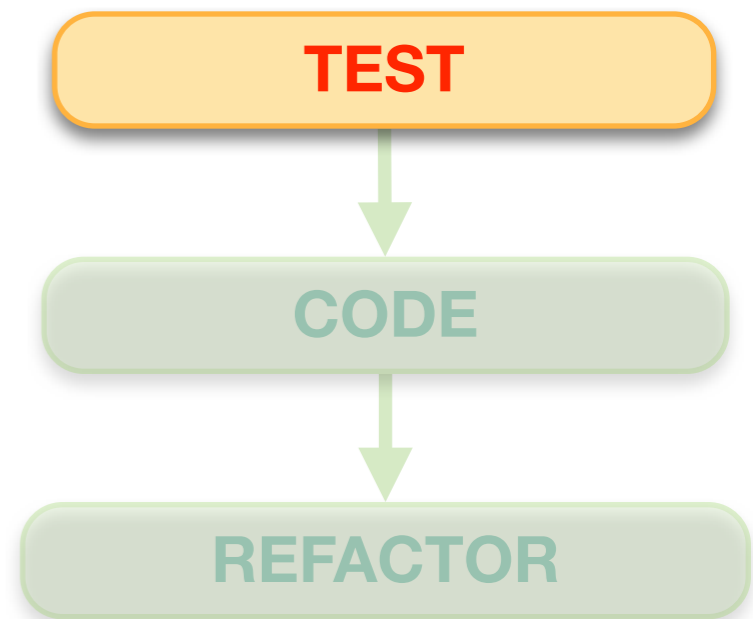
▶ The TDD development process

*“Never write a single line of code unless you have a failing automated test.
Eliminate duplication.”*

- Kent Beck

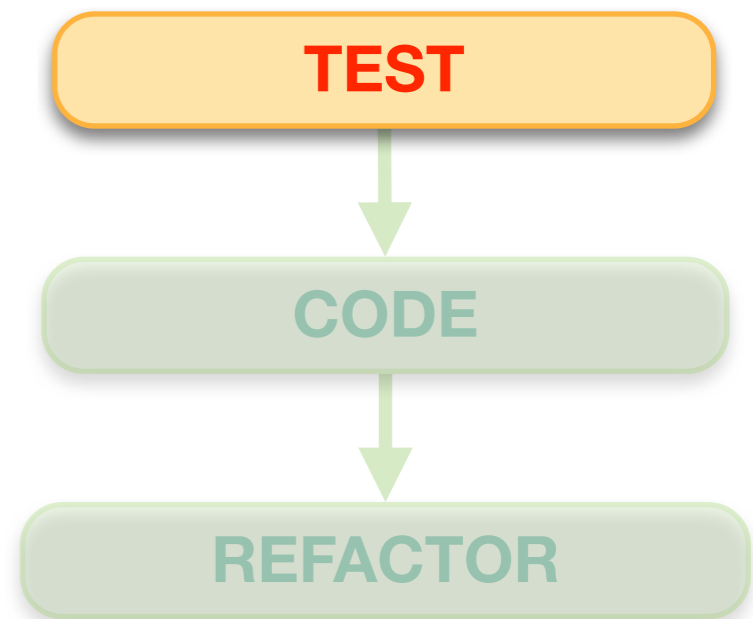


TDD Basics



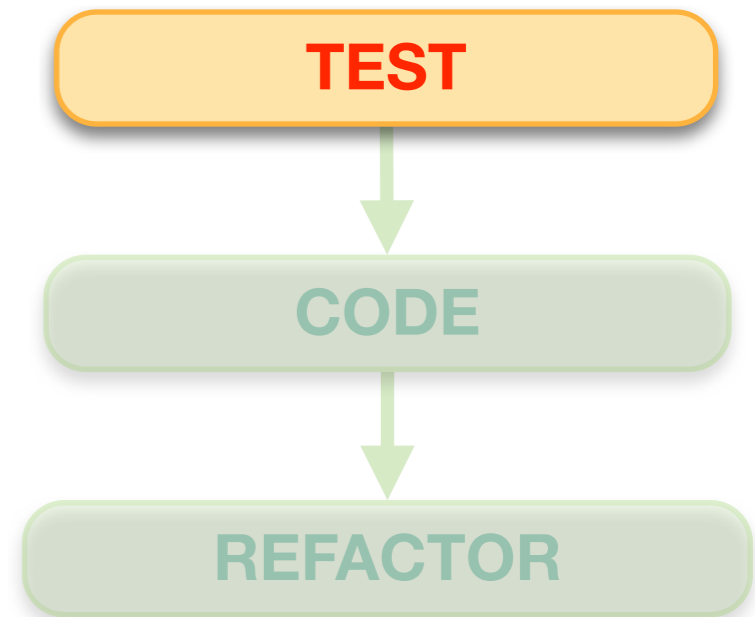
TDD Basics

▶ Step 1) Write a test



TDD Basics

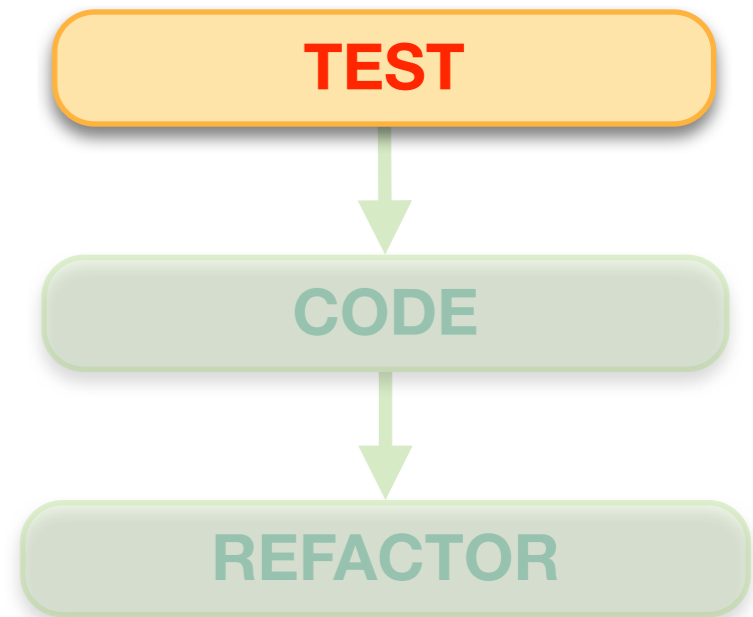
- ▶ **Step 1) Write a test**
- ▶ Design your code from the outside-in



TDD Basics

▶ Step 1) Write a test

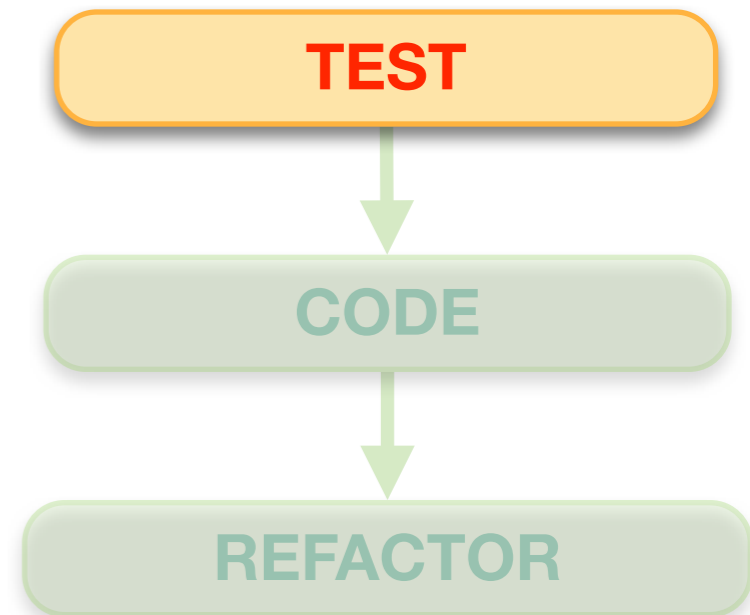
- ▶ Design your code from the outside-in
- ▶ How will your code be used?



TDD Basics

▶ Step 1) Write a test

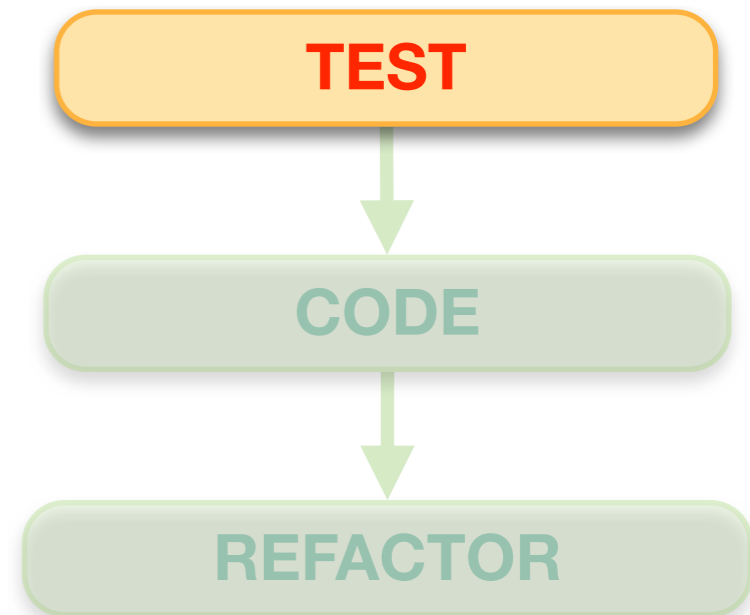
- ▶ Design your code from the outside-in
 - ▶ How will your code be used?
 - ▶ Design a clean interface for your code



TDD Basics

▶ Step 1) Write a test

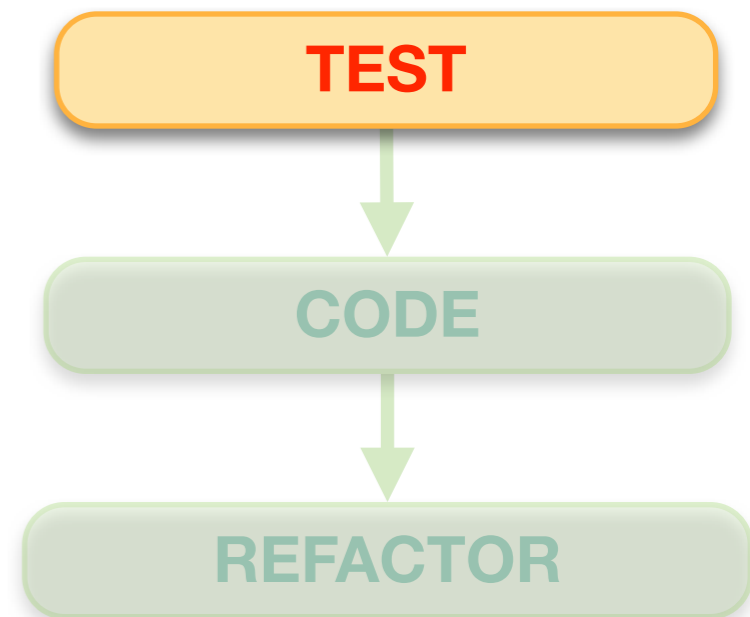
- ▶ Design your code from the outside-in
 - ▶ How will your code be used?
 - ▶ Design a clean interface for your code
 - ▶ Implement *only* features that are really needed



TDD Basics

▶ Step 1) Write a test

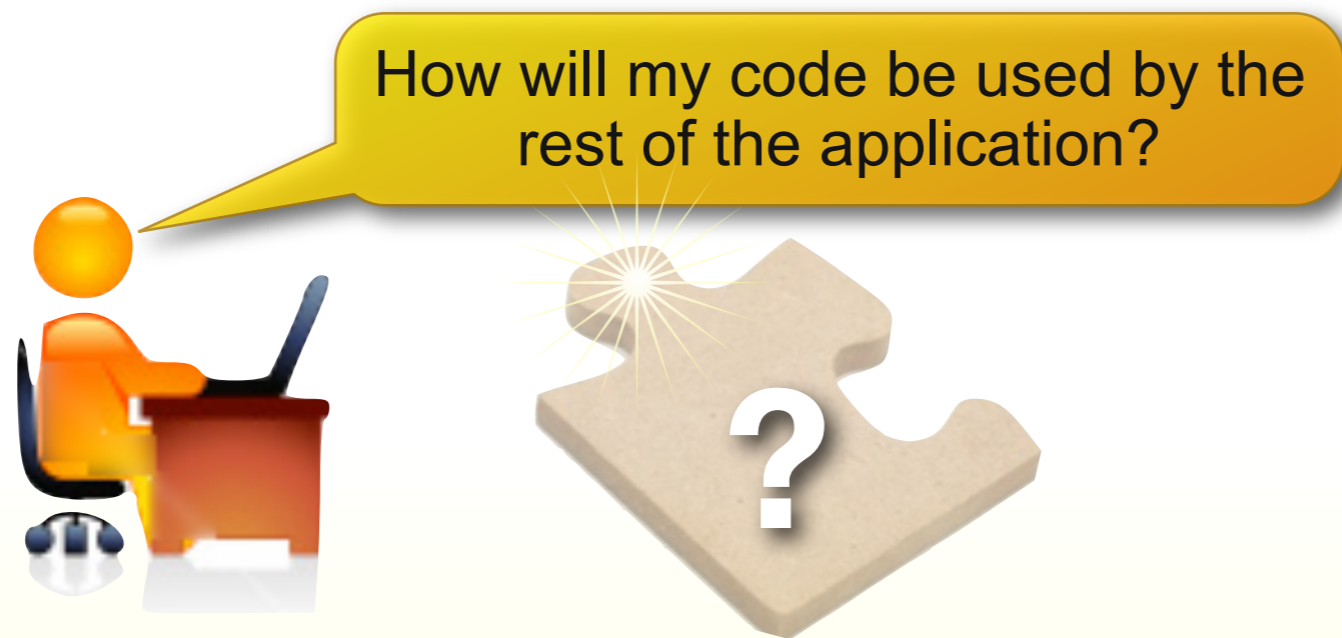
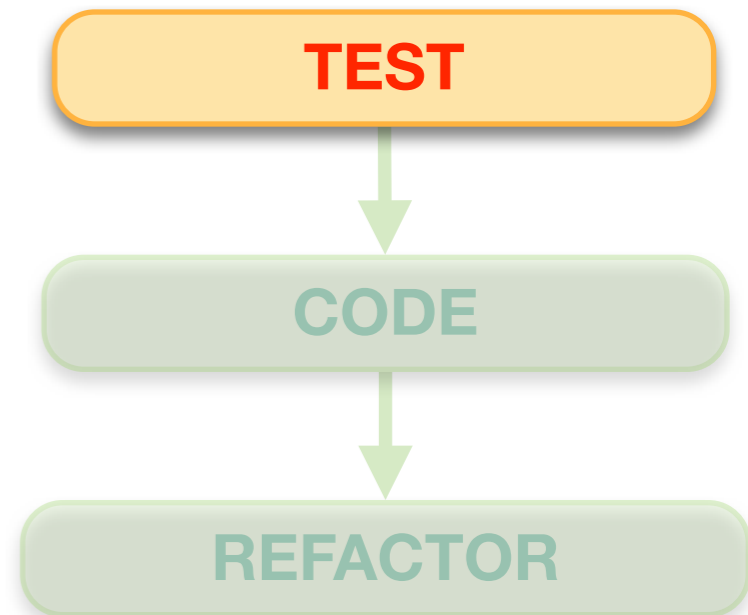
- ▶ Design your code from the outside-in
 - ▶ How will your code be used?
 - ▶ Design a clean interface for your code
 - ▶ Implement *only* features that are really needed
 - ▶ Provide examples of how you expect your code to be used



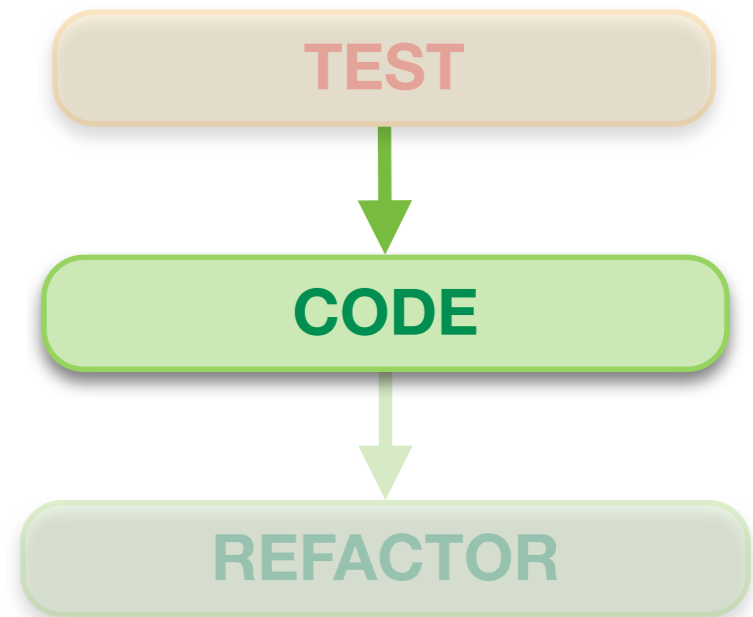
TDD Basics

▶ Step 1) Write a test

- ▶ Design your code from the outside-in
 - ▶ How will your code be used?
 - ▶ Design a clean interface for your code
 - ▶ Implement *only* features that are really needed
 - ▶ Provide examples of how you expect your code to be used

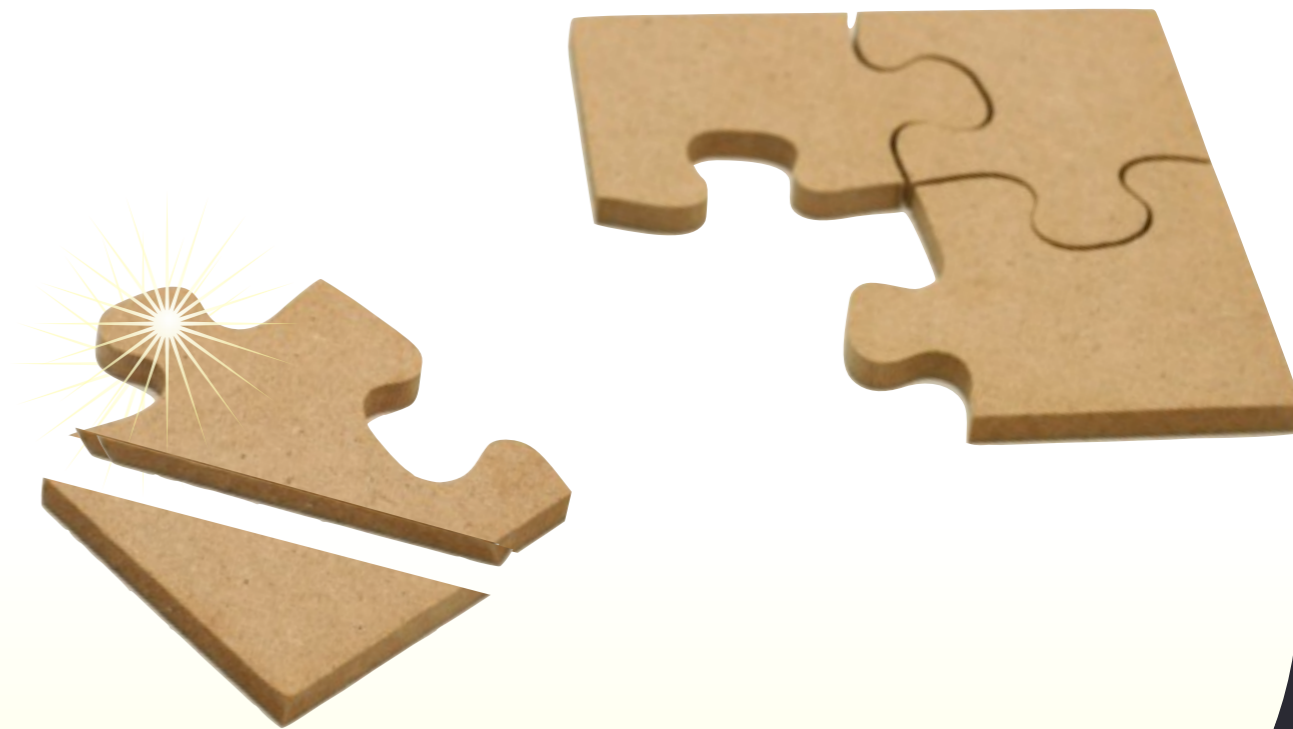
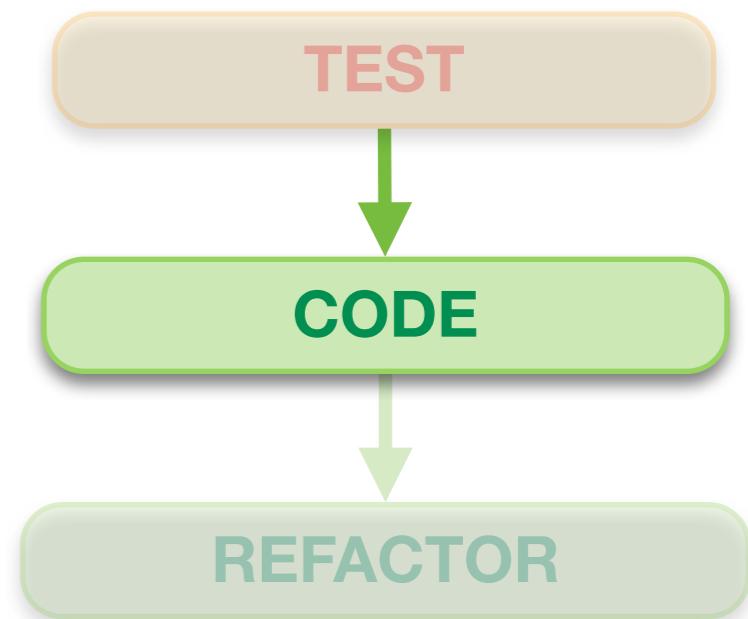


TDD Basics



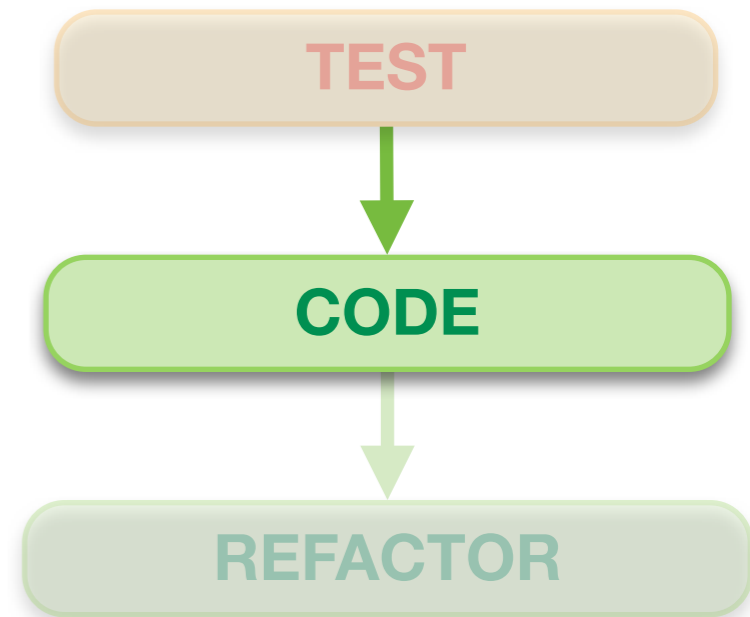
TDD Basics

▶ Step 2) Write some code



TDD Basics

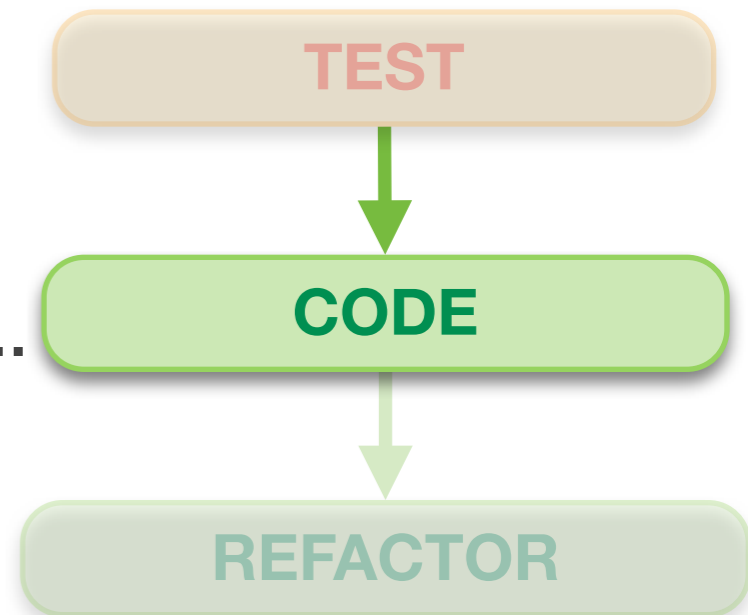
- ▶ **Step 2) Write some code**
 - ▶ Just enough code to make the tests pass



TDD Basics

▶ Step 2) Write some code

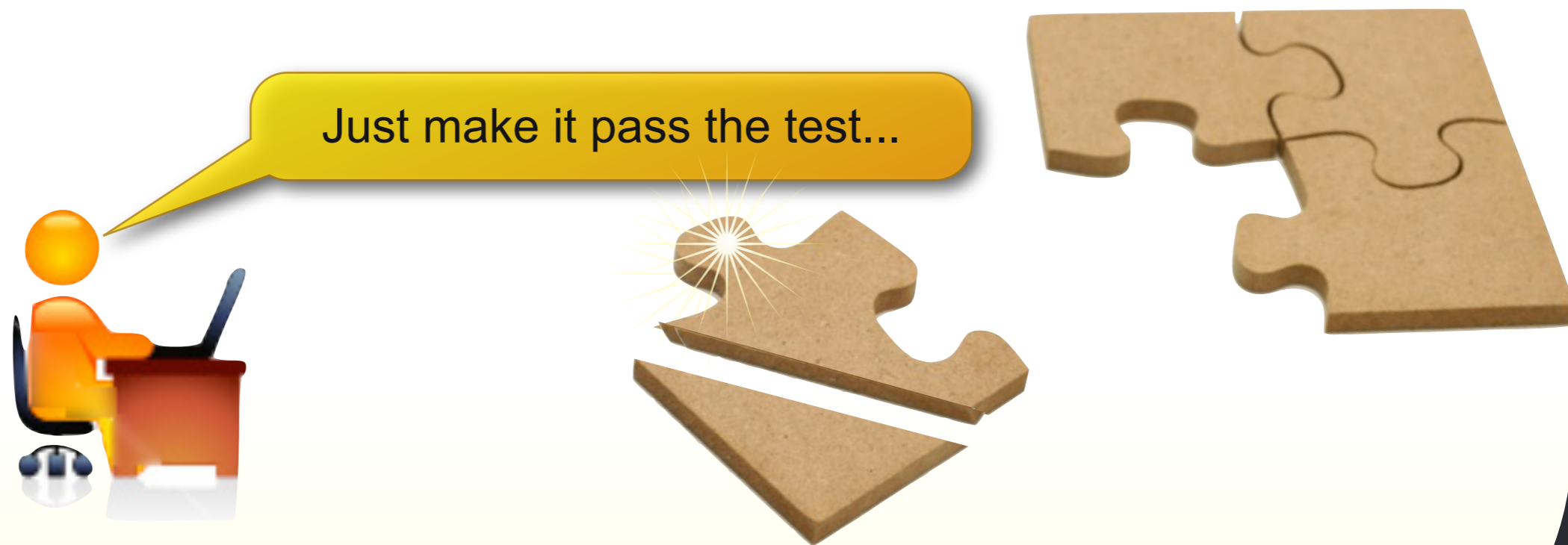
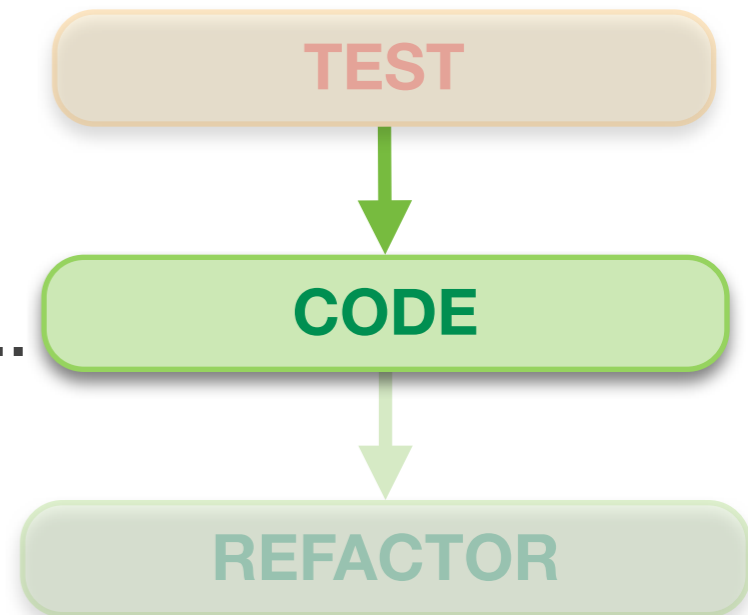
- ▶ Just enough code to make the tests pass
- ▶ Might mean a non-optimal solution at this stage...



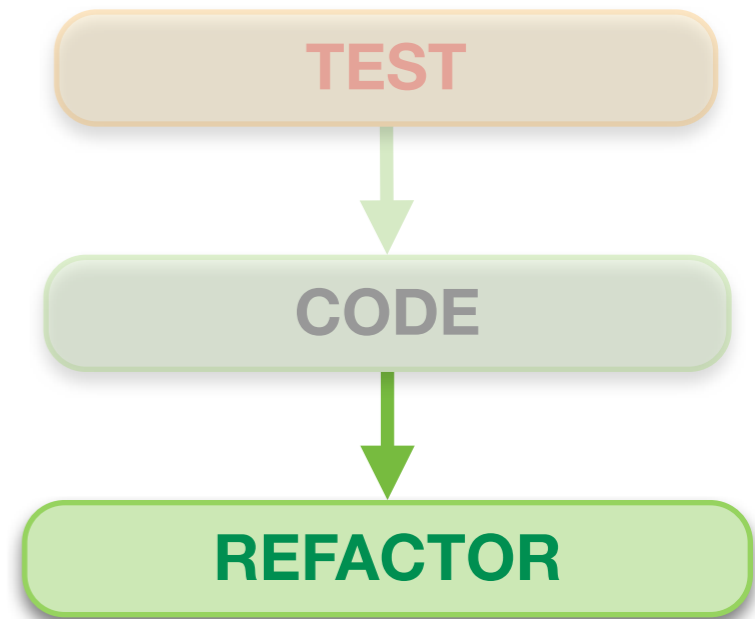
TDD Basics

▶ Step 2) Write some code

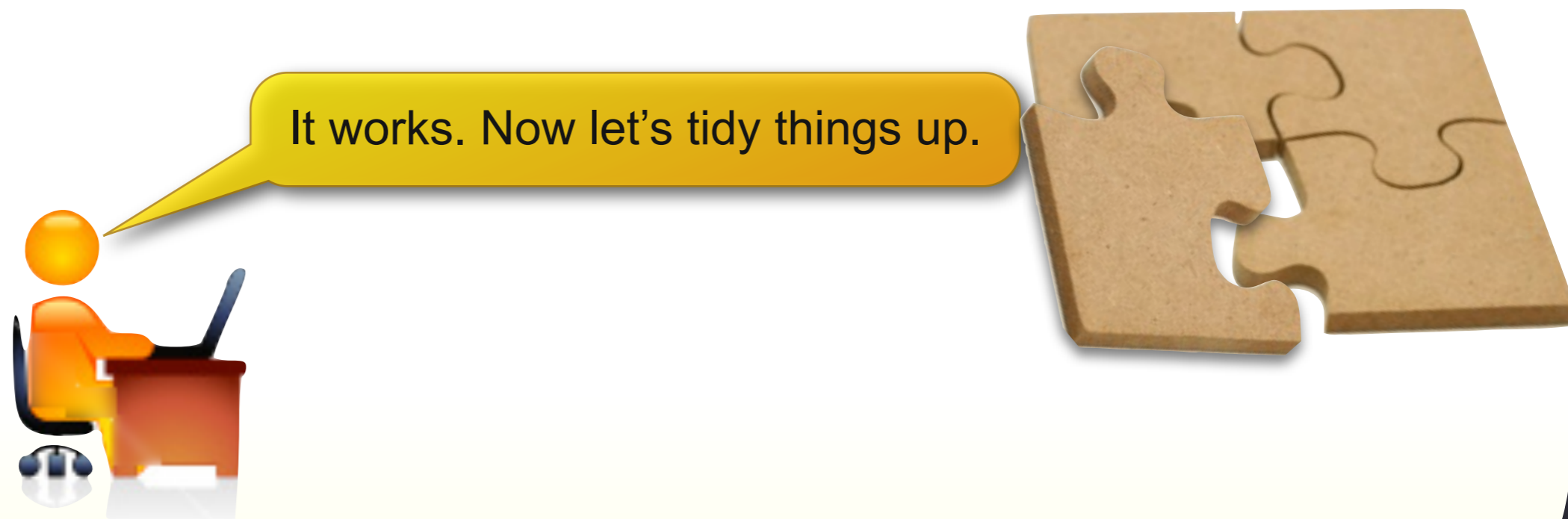
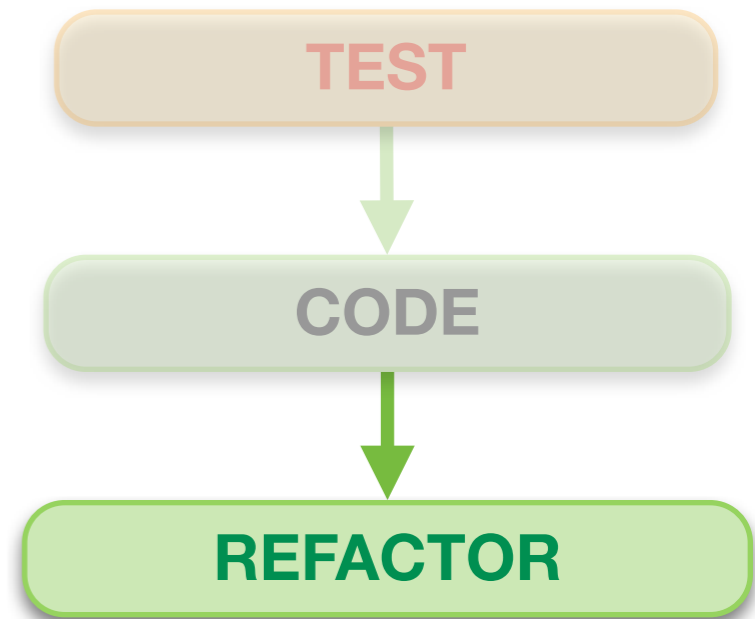
- ▶ Just enough code to make the tests pass
- ▶ Might mean a non-optimal solution at this stage...



TDD Basics

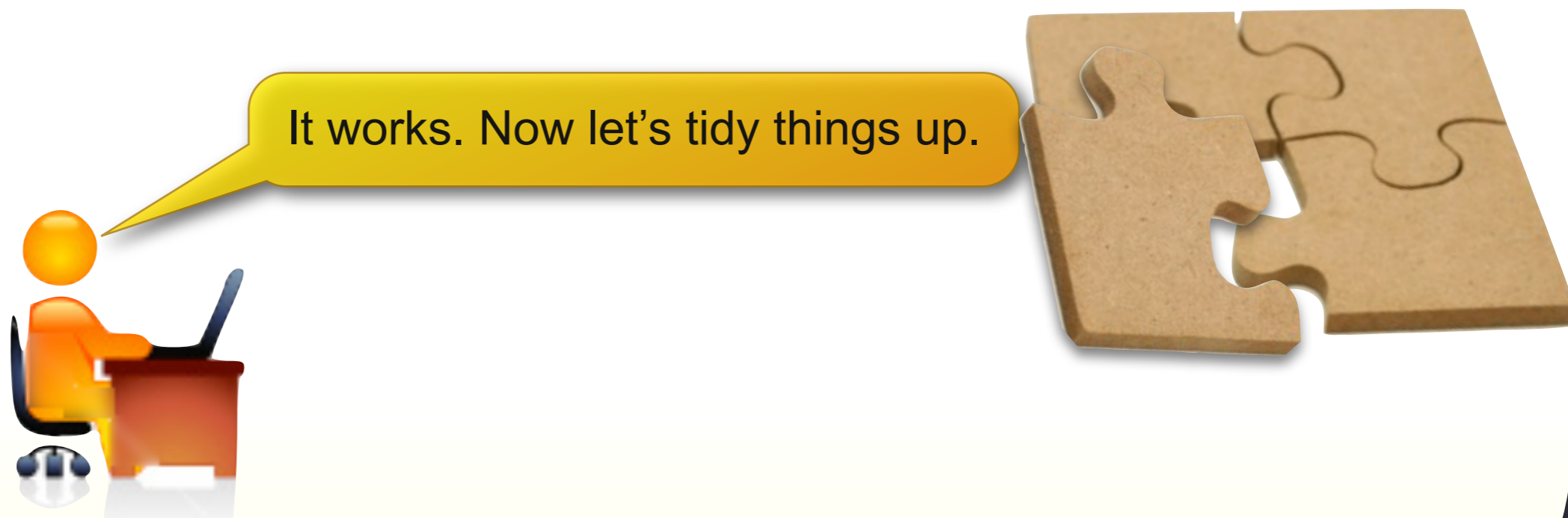
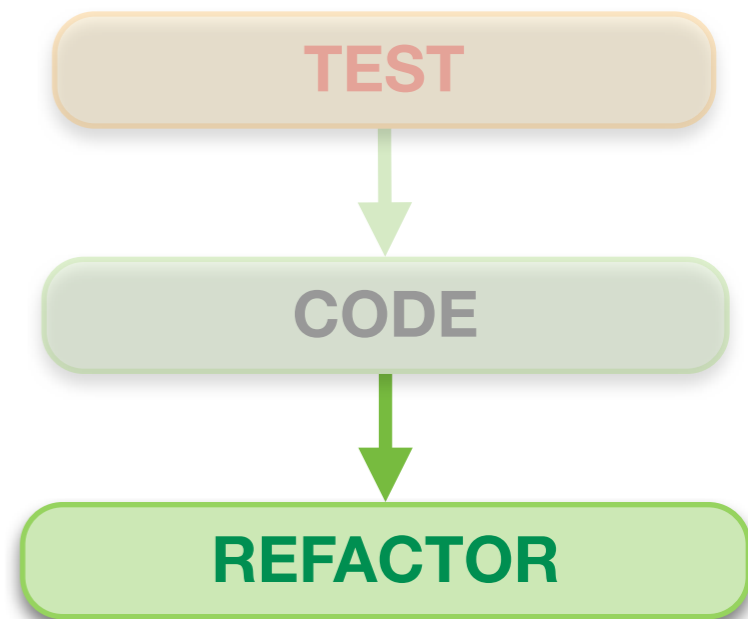


TDD Basics



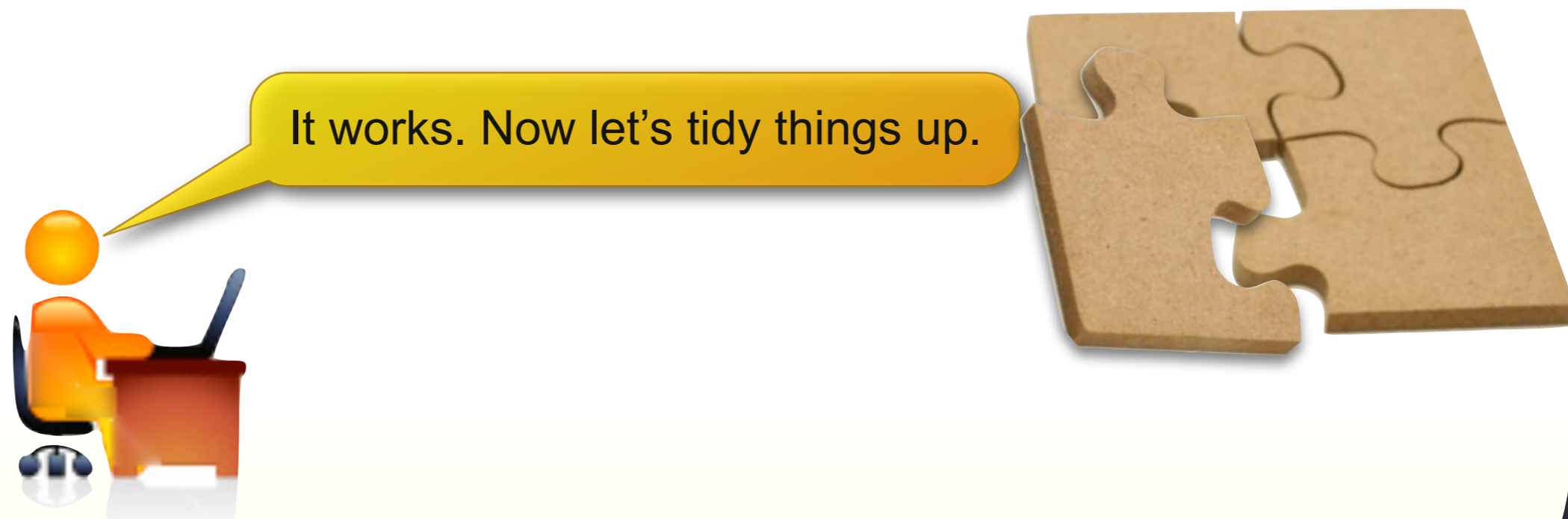
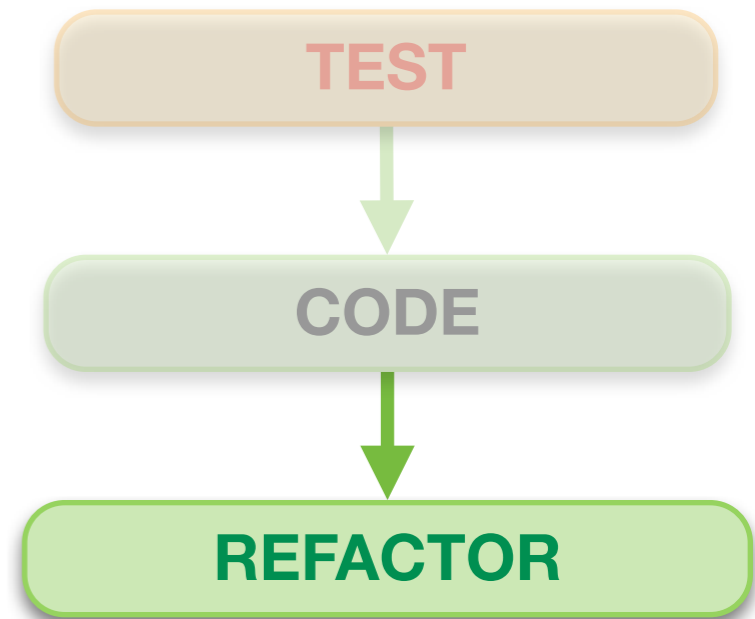
TDD Basics

▶ Step 3) Tidy up



TDD Basics

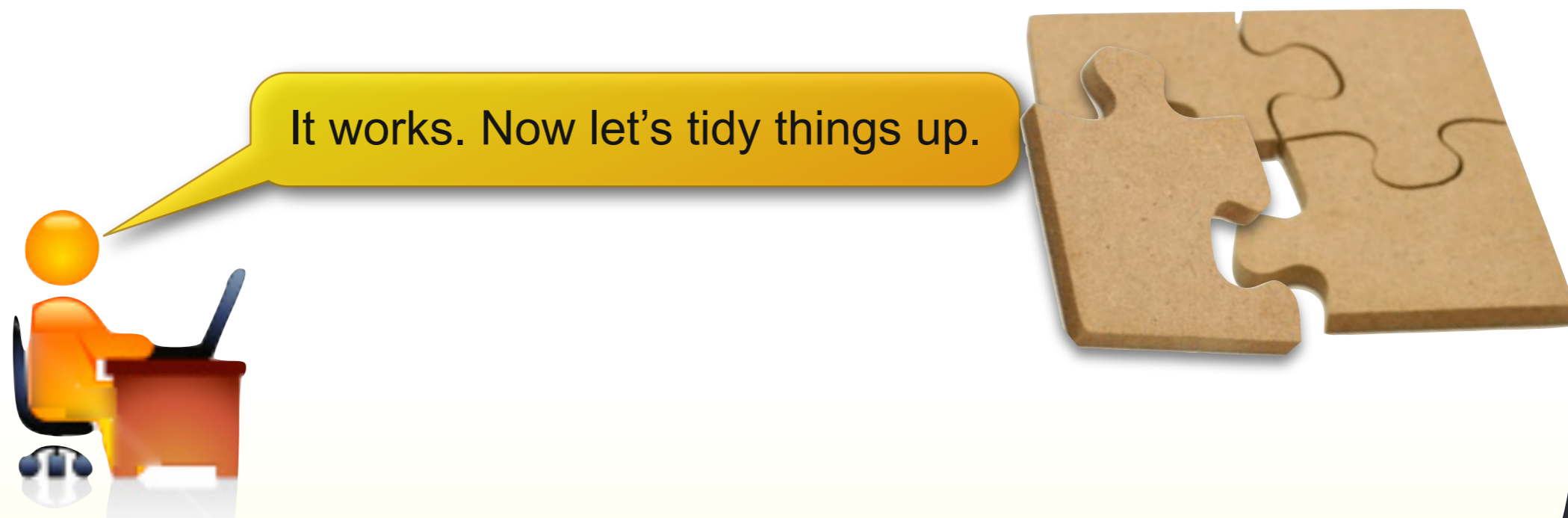
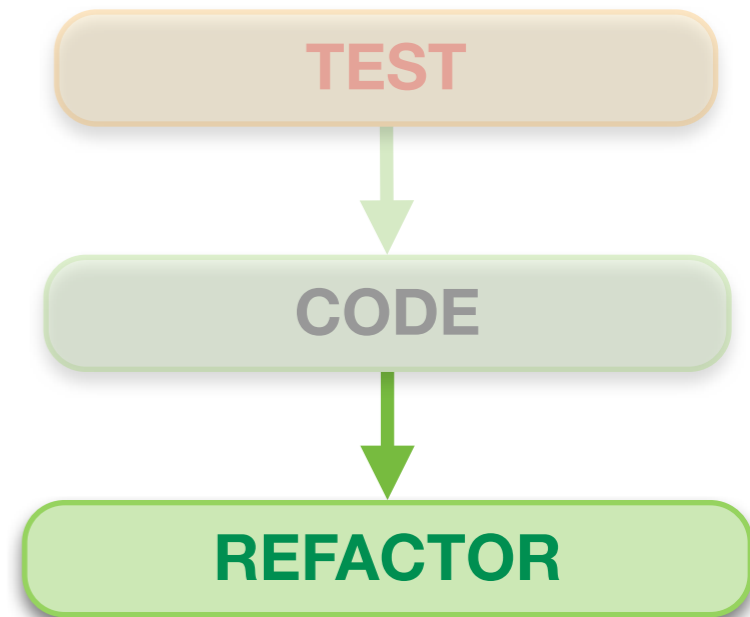
- ▶ **Step 3) Tidy up**
- ▶ Evolutionary (emerging) design



TDD Basics

▶ Step 3) Tidy up

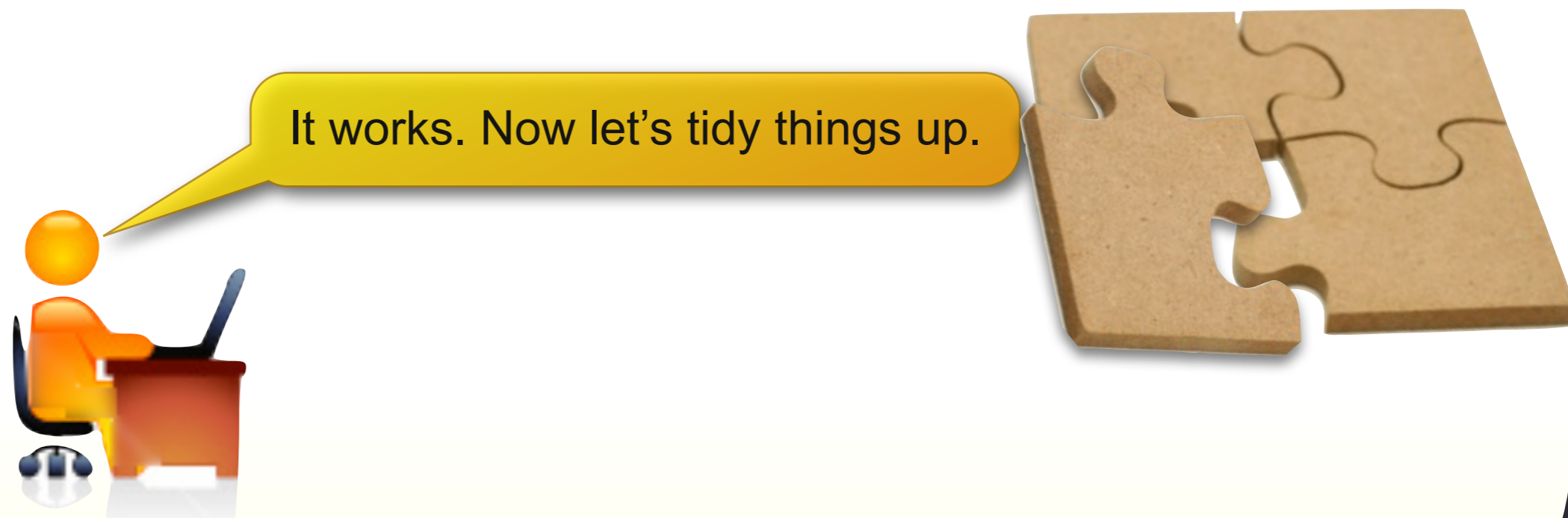
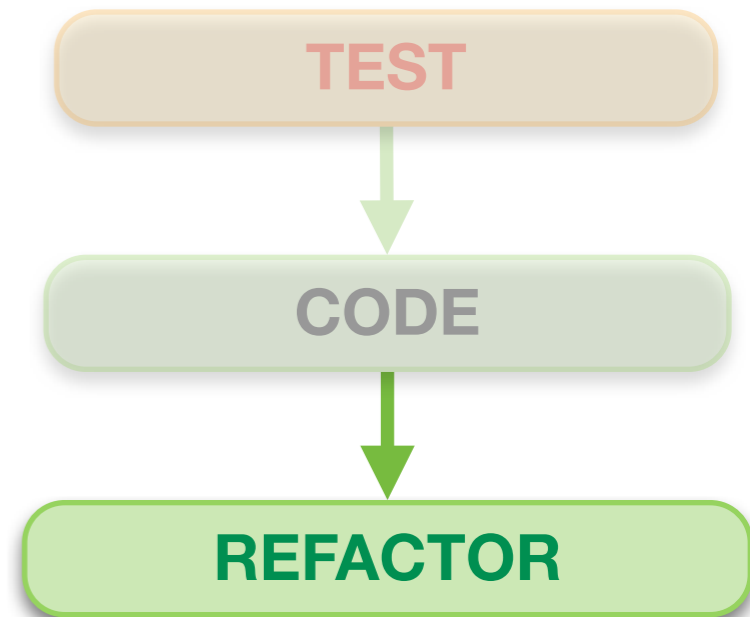
- ▶ Evolutionary (emerging) design
- ▶ Keep the code clean!



TDD Basics

▶ Step 3) Tidy up

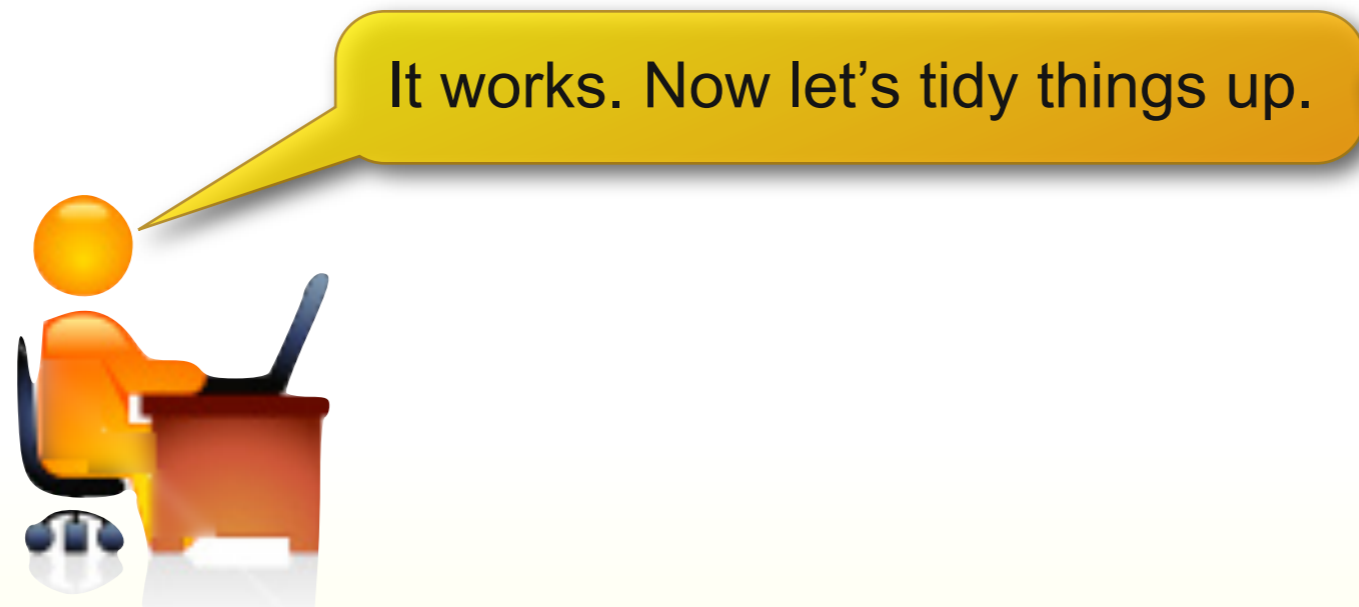
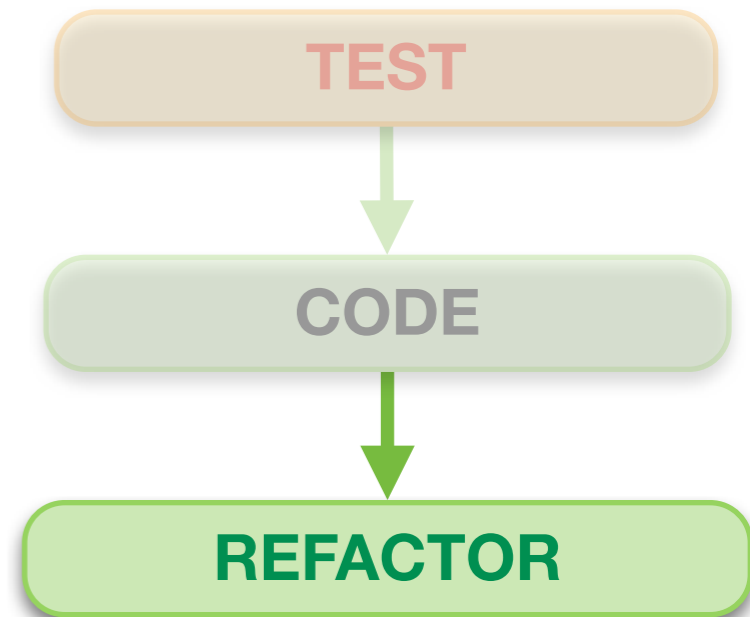
- ▶ Evolutionary (emerging) design
- ▶ Keep the code clean!
- ▶ Remove duplication and fix design problems



TDD Basics

▶ Step 3) Tidy up

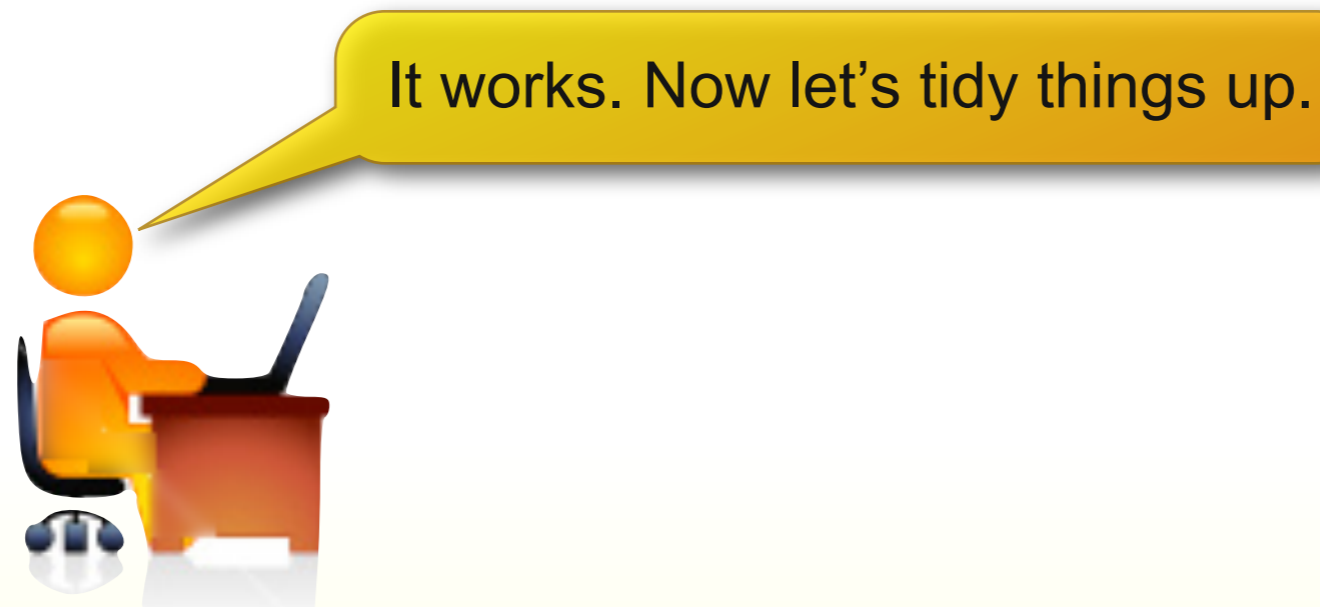
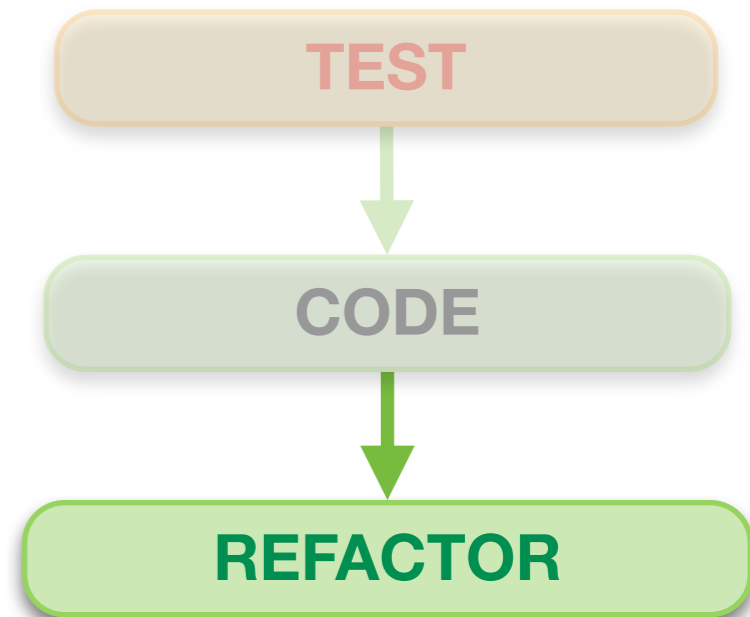
- ▶ Evolutionary (emerging) design
 - ▶ Keep the code clean!
 - ▶ Remove duplication and fix design problems
 - ▶ Restructure existing code - no new features!



TDD Basics

▶ Step 3) Tidy up

- ▶ Evolutionary (emerging) design
 - ▶ Keep the code clean!
 - ▶ Remove duplication and fix design problems
 - ▶ Restructure existing code - no new features!
 - ▶ Make sure the tests still pass!



TDD in Action - the Game Of Life



TDD in Action - the Game Of Life



So where do we start?

TDD in Action - the Game Of Life



So where do we start?

Let's list some requirements

TDD in Action - the Game Of Life

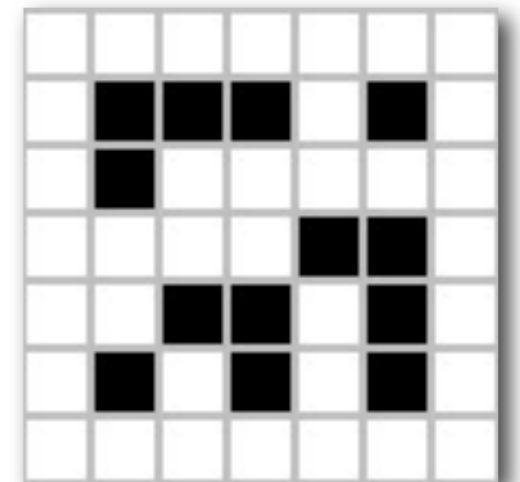


So where do we start?

Let's list some requirements

The Game of Life

- Live cells with less than 2 live neighbours die
- Live cells with more than 3 live neighbours die
- Live cells with 2 or 3 live neighbours remain alive
- Dead cells with 3 neighbours become alive



TDD in Action

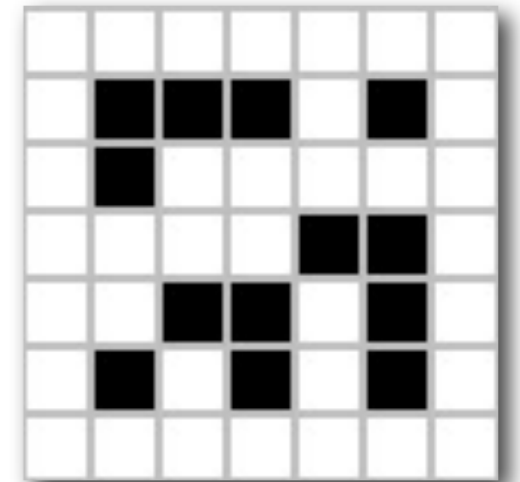


The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action



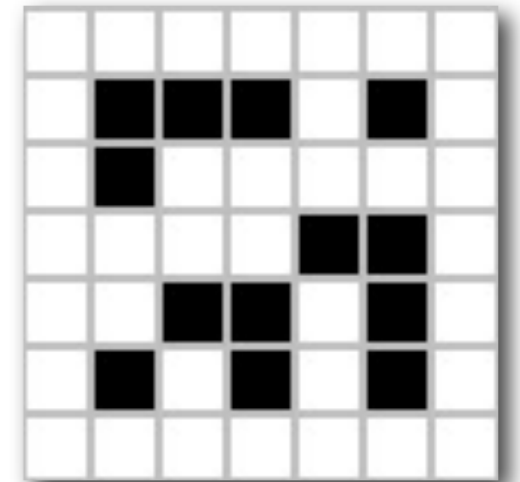
Now what?

The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action



Now what?

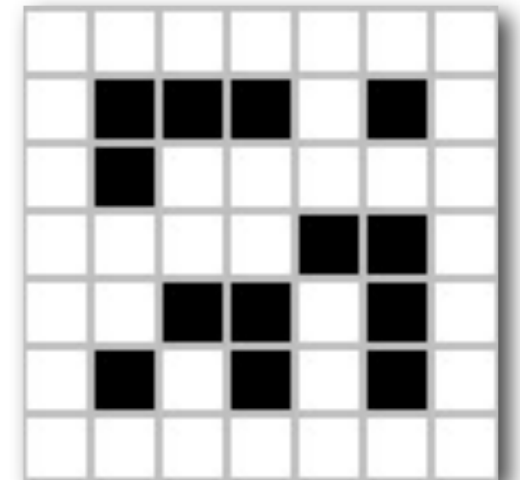
So what do we mean by this first requirement?

The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action

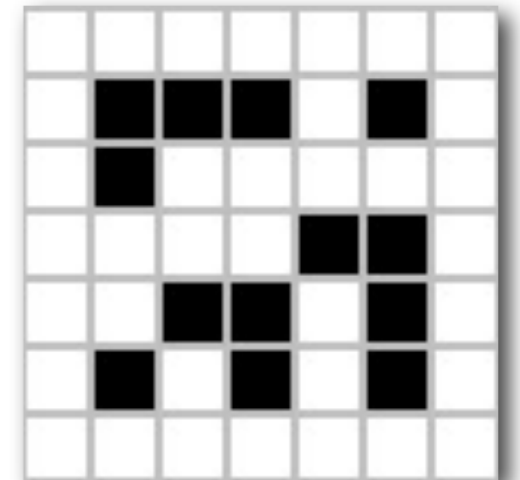


The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action



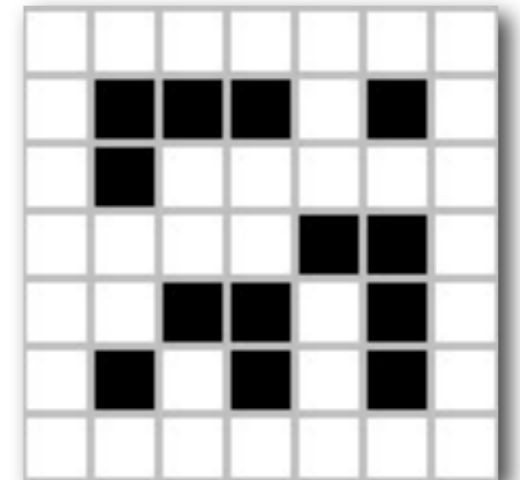
So what do we have to code for this one.

The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action



So what do we have to code for this one.

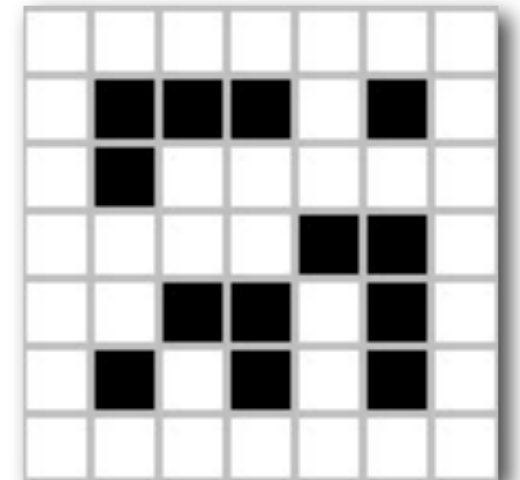
Dunno yet. Let's write some acceptance tests to find out

The Game of Life

- Live cells with less than 2 live neighbours die

User Story 1 - Less than 2 neighbours die

Given a cell with less than 2 live neighbours
When the next generation is created
Then this cell will die



TDD in Action



So what do we have to code for this one.

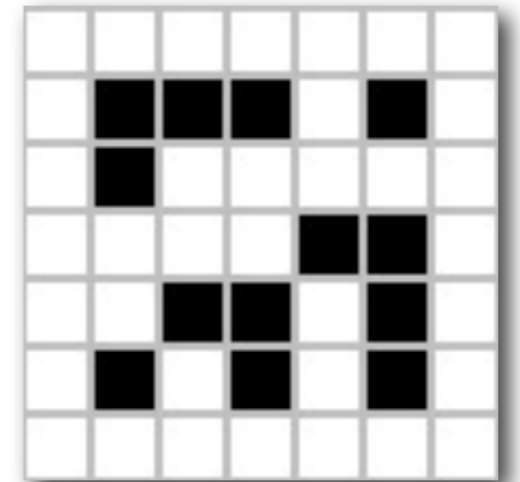
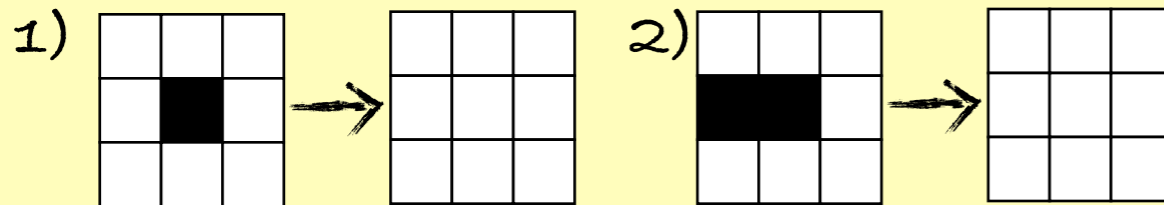
Dunno yet. Let's write some acceptance tests to find out

The Game of Life

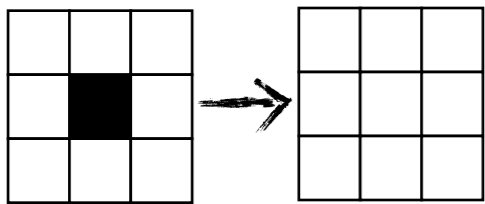
- Live cells with less than 2 live neighbours die

User Story 1 - Acceptance criteria

Sample grid transitions:



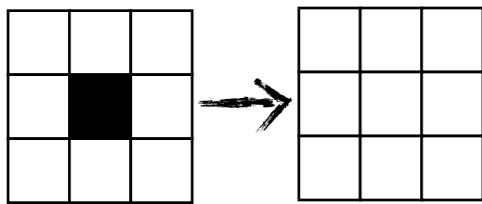
TDD in Action



TDD in Action



Great, so now I can start coding, right?

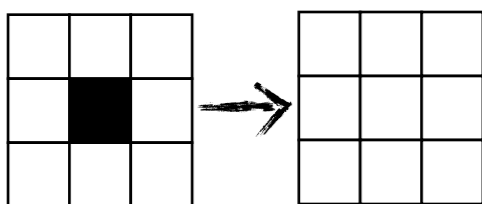


TDD in Action



Great, so now I can start coding, right?

Not so fast, hot shot. We need to turn this into an 'executable requirement'

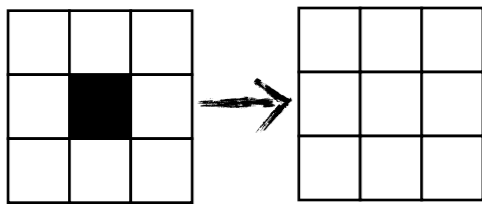


TDD in Action



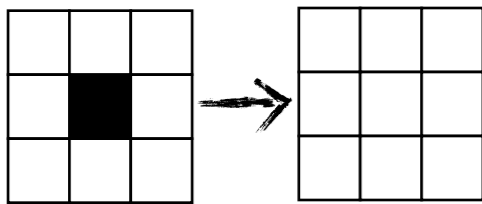
Great, so now I can start coding, right?

Not so fast, hot shot. We need to turn this into an 'executable requirement'



```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
    }  
}
```

TDD in Action

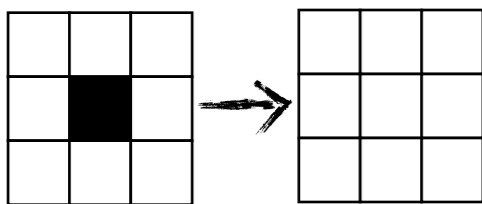


```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
    }  
}
```

TDD in Action



I still don't know what to write in this test...



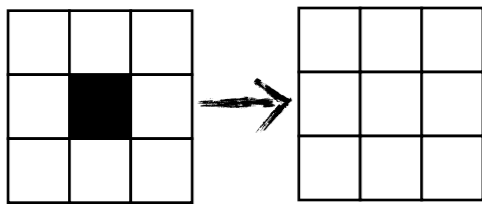
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
    }  
}
```

TDD in Action



I still don't know what to write in this test...

Start by expressing what you are trying to achieve



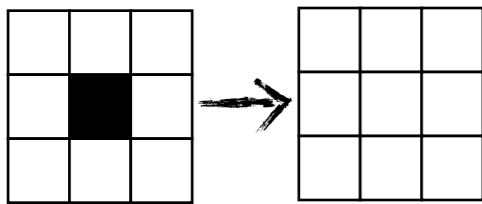
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
    }  
}
```

TDD in Action



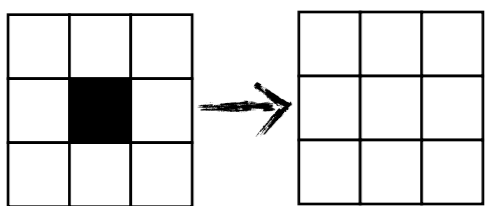
I still don't know what to write in this test...

Start by expressing what you are trying to achieve



```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String nextGrid = null;  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action

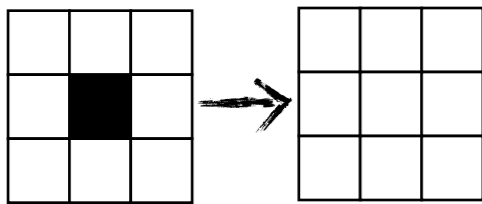


```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String nextGrid = null;  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



But that test will fail! What now?



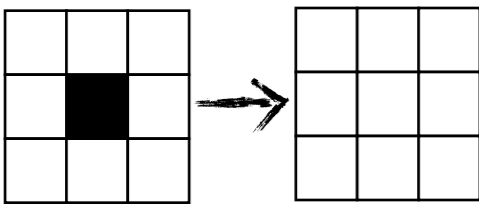
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String nextGrid = null;  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



But that test will fail! What now?

How you would like the code to look, if you had to use it?



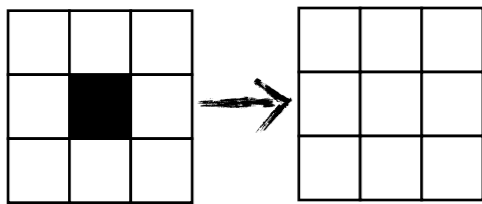
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String nextGrid = null;  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



But that test will fail! What now?

How you would like the code to look, if you had to use it?



```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
                               "...\\n" +  
                               "...";  
  
        String expectedNextGrid = "...\\n" +  
                                   "...\\n" +  
                                   "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So what's so special about this test?

```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So what's so special about this test?

The test name describes the expected behaviour

```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
                               "...\\n" +  
                               "...";  
  
        String expectedNextGrid = "...\\n" +  
                                   "...\\n" +  
                                   "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So what's so special about this test?

The test name describes the expected behaviour

```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

We started by expressing our expectations

TDD in Action



So what's so special about this test?

The test name describes the expected behaviour

```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
                               "...\\n" +  
                               "...";  
  
        String expectedNextGrid = "...\\n" +  
                                    "...\\n" +  
                                    "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

The code is written for API users, not API developers

We started by expressing our expectations

TDD in Action



So what's so special about this test?

The test name describes the expected behaviour

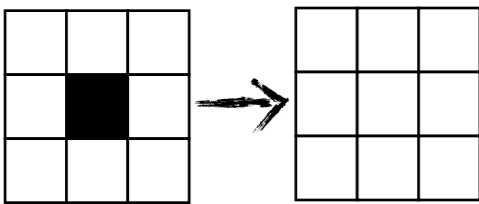
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

It's a working example of how to use the class

The code is written for API users, not API developers

We started by expressing our expectations

TDD in Action

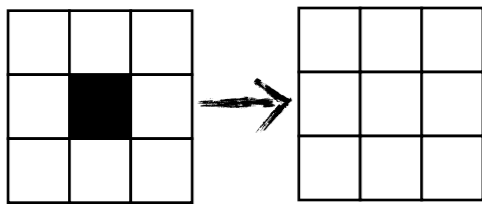


```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So what now?



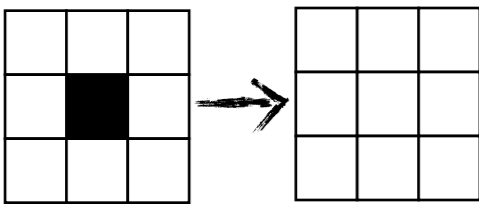
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So what now?

This is our acceptance test. Now we drill down.



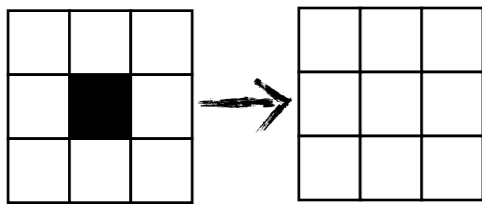
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



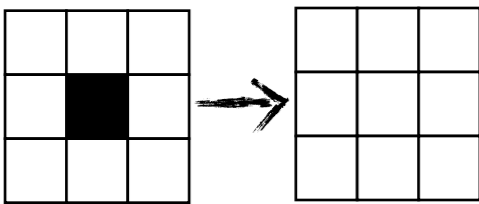
So what now?

This is our acceptance test. Now we drill down.



```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action

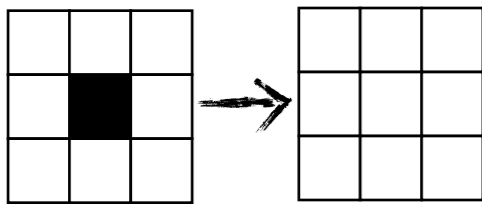


```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So I guess you'll say we need to write a test?



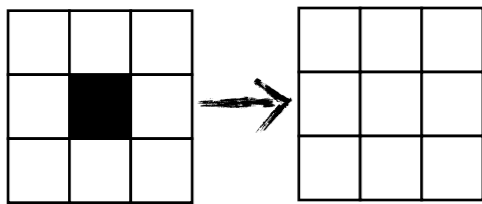
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



So I guess you'll say we need to write a test?

Spot on! You catch on fast!



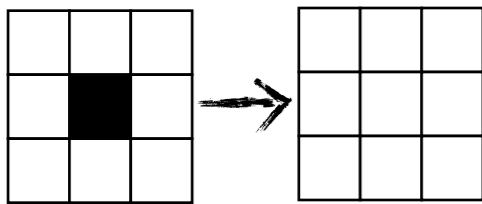
```
public class GameOfLifeTest {  
  
    @Test  
    public void aCellWithNoNeighboursShouldDieInTheNextGeneration() {  
        String initialGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        String expectedNextGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(initialGrid));  
        theUniverse.createNextGeneration();  
  
        String nextGrid = theUniverse.getGrid();  
        assertThat(nextGrid, is(expectedNextGrid));  
    }  
}
```

TDD in Action



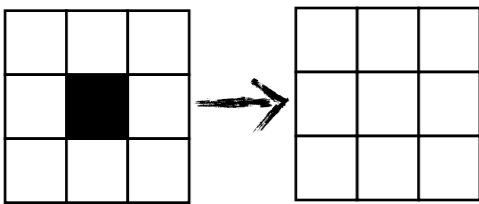
So I guess you'll say we need to write a test?

Spot on! You catch on fast!



```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmpyGrid() {  
  
        String seededGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}  
  
Universe theUniverse = new Universe(seededWith(initialGrid));  
theUniverse.createNextGeneration();  
  
String nextGrid = theUniverse.getGrid();  
assertThat(nextGrid, is(expectedNextGrid));  
}
```

TDD in Action

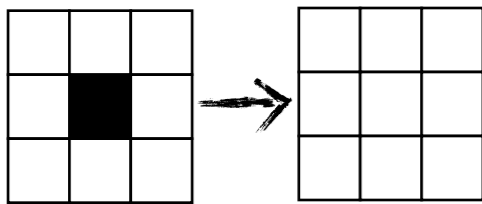


```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {  
  
        String seededGrid = "...\\n" +  
                            "...\\n" +  
                            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}
```

TDD in Action



So now can we code?



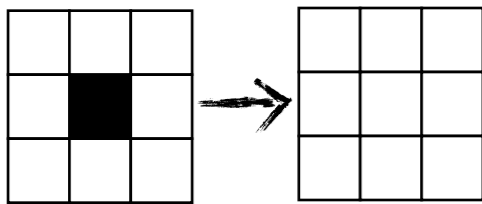
```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {  
  
        String seededGrid = "...\\n" +  
            "...\\n" +  
            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}
```

TDD in Action



So now can we code?

Yes, but just a little bit



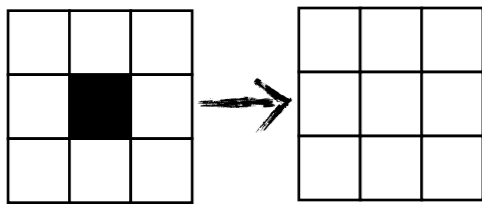
```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {  
  
        String seededGrid = "...\\n" +  
                            "...\\n" +  
                            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}
```

TDD in Action



So now can we code?

Yes, but just a little bit



```
public class UniverseTest {
```

```
    @Test  
    public void aUniver
```

```
        String seededGr
```

```
        Universe theUni  
        String currentG  
        assertThat(curr
```

```
    }  
}
```

```
public class Universe {
```

```
    public Universe(String initialGridContents) {}
```

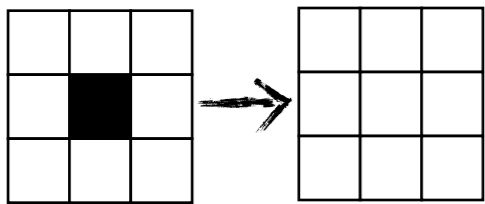
```
    public static String seededWith(String gridContents) {  
        return null;  
    }
```

```
    public void createNextGeneration() {}
```

```
    public String getGrid() {  
        return "...\\n" +  
            "...\\n" +  
            "...";  
    }
```

```
}
```

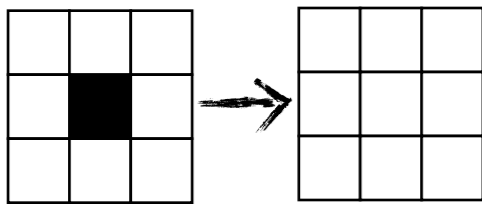
TDD in Action



TDD in Action



Er, isn't that a bit *too* simple.

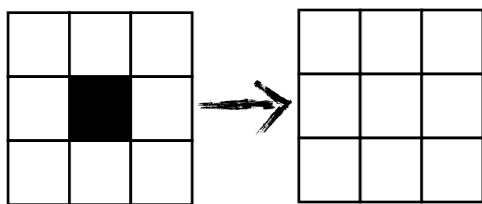


TDD in Action



Er, isn't that a bit *too* simple.

Yes. We need another test to tease out a more complete design

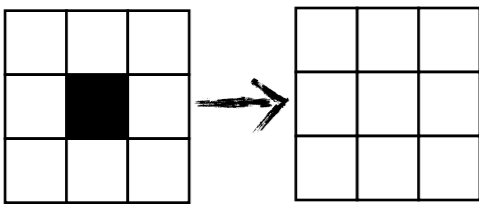


TDD in Action



Er, isn't that a bit *too* simple.

Yes. We need another test to tease out a more complete design



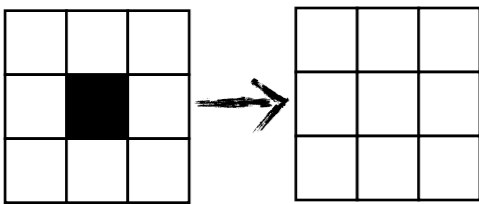
```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {...}  
  
    @Test  
    public void aUniverseSeededWithANonEmpyGridContentWillContainThatGrid() {  
  
        String seededGrid = "...\\n" +  
                            ".*.\\n" +  
                            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}
```

TDD in Action



Er, isn't that a bit *too* simple.

Yes. We need another test to tease out a more complete design



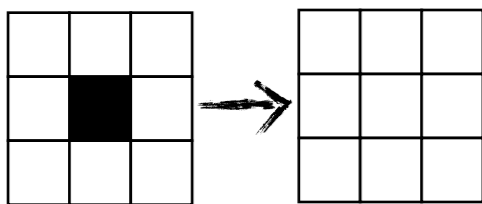
```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {...}  
  
    @Test  
    public void aUniverseSeededWithANonEmpyGridContentWillContainThatGrid() {  
  
        String seededGrid = "...\\n" +  
                            ".*.\\n" +  
                            "...";  
  
        Universe theUniverse = new Universe(seededWith(seededGrid));  
        String currentGrid = theUniverse.getGrid();  
        assertThat(currentGrid, is(seededGrid));  
    }  
}
```

TDD in Action



Er, isn't that a bit *too* simple.

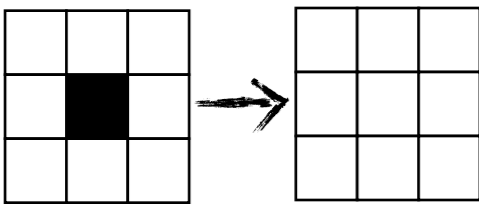
Yes. We need another test to tease out a more complete design



```
public class UniverseTest {  
  
    @Test  
    public void aUniverseSee  
  
    @Test  
    public void aUniverseSee  
  
        String seededGrid =  
  
        Universe theUniverse  
        String currentGrid =  
        assertThat(currentGr  
  
    }  
}
```

```
public class Universe {  
  
    private String currentContent;  
  
    public Universe(String initialGridContents) {  
        currentContent = initialGridContents;  
    }  
  
    public static String seededWith(String gridContents) {  
        return gridContents;  
    }  
  
    public void createNextGeneration() {  
    }  
  
    public String getGrid() {  
        return currentContent;  
    }  
}
```

TDD in Action

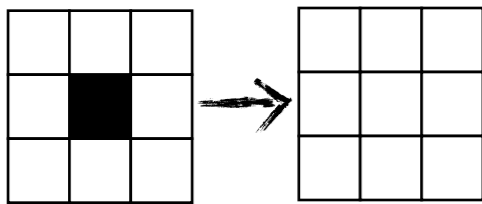


```
public class Universe {  
  
    private String currentContent;  
  
    public Universe(String initialGridContents) {  
        currentContent = initialGridContents;  
    }  
  
    public static String seededWith(String gridContents) {  
        return gridContents;  
    }  
  
    public void createNextGeneration() {  
    }  
  
    public String getGrid() {  
        return currentContent;  
    }  
}
```

TDD in Action



Well, now at least those two tests pass.



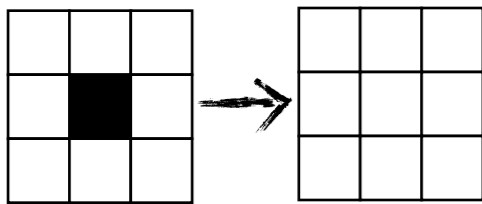
```
public class Universe {  
  
    private String currentContent;  
  
    public Universe(String initialGridContents) {  
        currentContent = initialGridContents;  
    }  
  
    public static String seededWith(String gridContents) {  
        return gridContents;  
    }  
  
    public void createNextGeneration() {  
    }  
  
    public String getGrid() {  
        return currentContent;  
    }  
}
```

TDD in Action



Well, now at least those two tests pass.

Now we need to implement the 'createNextGeneration()' method...



```
public class Universe {  
  
    private String currentContent;  
  
    public Universe(String initialGridContents) {  
        currentContent = initialGridContents;  
    }  
  
    public static String seededWith(String gridContents) {  
        return gridContents;  
    }  
  
    public void createNextGeneration() {  
    }  
  
    public String getGrid() {  
        return currentContent;  
    }  
}
```

TDD for unbelievers



So TDD is the answer to all my problems?

No, sorry, TDD is not a Silver Bullet



- ▶ You still have to use your brain
 - ▶ Also use other design activities (domain modeling, DDD,...)
- ▶ TDD is not applicable everywhere
 - ▶ Highly visual coding, ‘know it when you see it’ stuff,...
- ▶ TDD works best if you have a vision

TDD for unbelievers



But how can I write tests if I don't know what the code looks like?

How can you write code when you can't express what it should do?

“Write code for others as you would have them write code for you”

- Some agile dude

- ▶ TDD is about expressing the intent of your code
- ▶ It is a design practice
- ▶ Design your code “outside-in”

TDD for unbelievers



I don't have time to write tests

But you have time to fix the code afterwards, right?

“I can get it done much faster if it doesn't have to work”

- Kent Beck (paraphrased)

- ▶ New TDDers will take maybe 20-30% longer
- ▶ Experienced TDD developers don't spend much (if any) more time coding
- ▶ But the code is of much higher quality

TDD for unbelievers



But I'm writing more test code than application code!

Sure, writing tests is part of the process

- ▶ You will write lots of tests...
- ▶ ...but your code will be more focused and more accurate

TDD for unbelievers



My tests break whenever I change my code

Try to test the behaviour, not the implementation

- ▶ Validate behaviour, don't verify implementation

TDD for unbelievers



Our tests are too hard and take too much time to keep up to date

Treat your test code as production code

- ▶ Be wary of test maintenance overhead
 - ▶ Test code is not second-class code
 - ▶ Refactor to keep your test code maintainable
 - ▶ Avoid technical debt

TDD for unbelievers



I found a bug in the production code. TDD doesn't work.

see "TDD is not a Silver Bullet"

- ▶ You still need your testers!
 - ▶ You can still make mistakes...
 - ▶ ...but they are easier to isolate and to fix
 - ▶ If you find a bug, just add another test to reproduce it, then fix it!

TDD for unbelievers



I've heard TDD encourages sloppy design. Isn't it better to have a general vision of the problem before coding the solution?"

Sometimes, yes. That's what the refactoring phase is for.

- ▶ TDD does not preclude initial high-level design
 - ▶ Brainstorm a high-level design approach
 - ▶ Define a common well-known domain language
 - ▶ Use an 'Iteration 0' and spikes
 - ▶ Don't be afraid to refactor
- ▶ TDD works better when you have a vision

TDD for unbelievers



I use a database/network/web service... and TDD-style unit tests can't test it. TDD doesn't work

You will need integration tests too

- ▶ Unit tests when you can, integrations tests when you must
 - ▶ Integration tests for system boundaries
 - ▶ Mock out these boundary classes for the rest of your tests

TDD for unbelievers

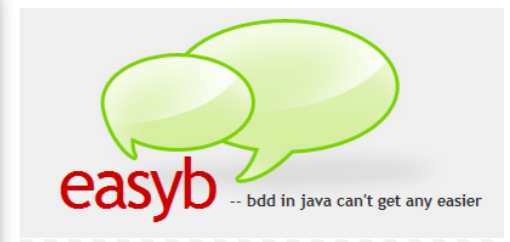


Our BAs and product owners aren't buying this TDD stuff

Try putting them into the feedback loop

- ▶ Acceptance-Test Driven Development is about communication
- ▶ BDD help BAs and POs participate in writing executable requirements

```
scenario "A cell with less than 2 live neighbours should die", {  
  given "Given a cell with no live neighbours",  
  when "a new generation is generated",  
  then "that cell will have died"  
}
```



TDD for unbelievers



Our BAs and product owners aren't buying this TDD stuff

Try putting them into the feedback loop

- ▶ Acceptance-Test Driven Development is about communication
- ▶ BDD help BAs and POs participate in writing executable requirements

```
scenario "A cell with less than 2 live neighbours should die", {
  given "Given a cell with no live neighbours", {
    initialGrid = ""...
                  .*
                  ...""
    theUniverse = new Universe(seededWith(initialGrid));
  }
  when "a new generation is generated", {
    theUniverse.createNextGeneration()
  }
  then "that cell will have died", {
    theUniverse.grid.shouldBe ""...
                              ...
                              ...""
  }
}
```



TDD for unbelievers



Our BAs and product owners aren't buying this TDD stuff

Try putting them into the feedback loop

- ▶ Acceptance-Test Driven Development is about communication
- ▶ BDD help BAs and POs participate in writing executable requirements

```
Given a living cell with no live neighbours like this
```

```
...
```

```
.*
```

```
...
```

```
When a new generation is generated  
then the grid should look like this:
```

```
...
```

```
...
```

```
...
```



TDD for unbelievers

Many BDD tools also have great reporting features



easyb -- bdd in java can't get any easier

sections

- Summary
- Stories
- Stories Text

Summary

Behaviors	Failed	Pending	Time (s)
4	1	1	0.738

Stories Summary

Stories	Scenarios	Failed	Pending	Time (sec)
2	4	1	1	0.738

Specifications Summary

Specifications	Failed	Pending	Time (sec)
0	0	0	0.0

TDD for unbelievers

Many BDD tools have great reporting features

Test results summary

The screenshot displays an 'easyb-report' window with a 'Stories List' table and detailed test results for a 'Withdraw money from an account' story. The 'Withdraw more money than there is in the account' scenario is highlighted as a failure.

Story	Scenarios	Failed	Pending	Time (sec)
create account	1	0	0	0.608
withdraw money from account	3	1	1	0.121

Story	Scenario	Result	Time (sec)
Withdraw money from an account	Withdraw money from an account	success	0.0050
	given an account with a certain balance		
	when a sum is withdrawn from the account		
Withdraw more money than there is in the account	Withdraw more money than there is in the account	failure	0.026
	given an account with a certain balance		
	when an amount is withdrawn that is greater than the balance		
Withdraw money from blocked account	Withdraw money from blocked account	pending	
	given a blocked account		
	when money is withdrawn from the account		

Test failure details:
expected 150 but was 100
sun.reflect.NativeConstructorAccessorImpl.newInstance(...)
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructor...)
java.lang.reflect.Constructor.newInstance(Constructor.java:494)
org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:77)
org.codehaus.groovy.reflection.CachedConstructor.doConstructorInvoke(CachedConstructor.java:71)
org.codehaus.groovy.runtime.callsite.ConstructorSite\$ConstructorSiteNoUnwrap.callConstructor(ConstructorSite.java:84)
org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:52)
org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:192)
org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:200)
org.easyb.BehaviorCategory.throwValidationException(BehaviorCategory.groovy:171)

Test failure details

Unimplemented stories

TDD for unbelievers




“I’ve tried TDD, but I soon lapse back into my old code-first habits”

TDD takes practice and know-how

- ▶ Unit tests when you can, integrations tests when you must
 - ▶ Learn about TDD - read books on TDD practices
 - ▶ Do pair program with someone who knows TDD
 - ▶ Participate in and organize coding dojos
 - ▶ `<blatant plug>Training/mentoring</blatant plug>`

Conclusion

▶ Real Developers Don't Need Unit Tests?



So maybe Chuck should be writing unit tests after all...

I agree. You go talk to him



Thank You



John Ferguson Smart
Wakaleo Consulting Ltd.

<http://www.wakaleo.com>

Email: john.smart@wakaleo.com

Twitter: [wakaleo](https://twitter.com/wakaleo)