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

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

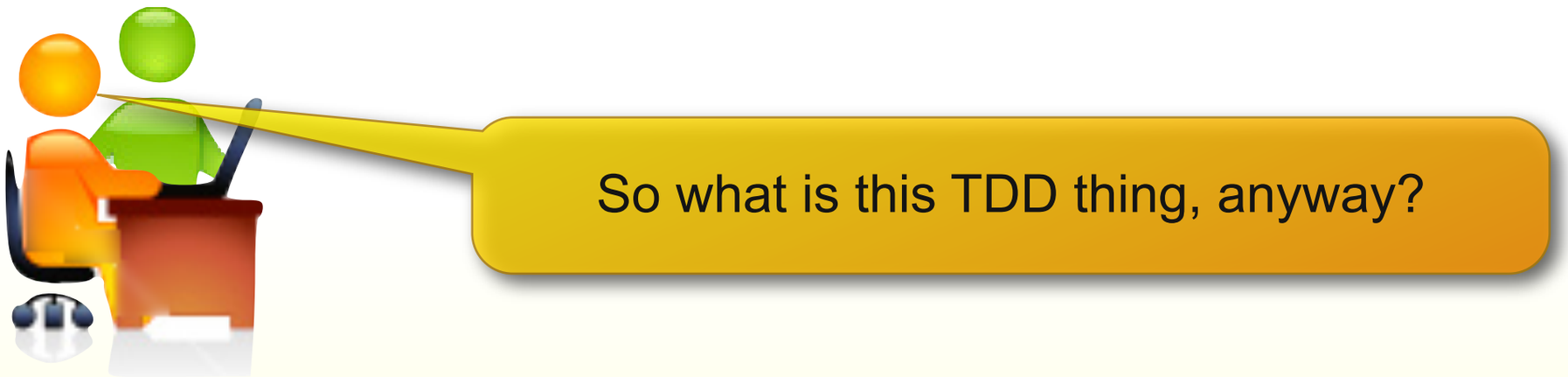
- Real Developers Don’t Need Unit Tests

OK, so Chuck Norris doesn’t need Unit Tests

But what about the rest of us?

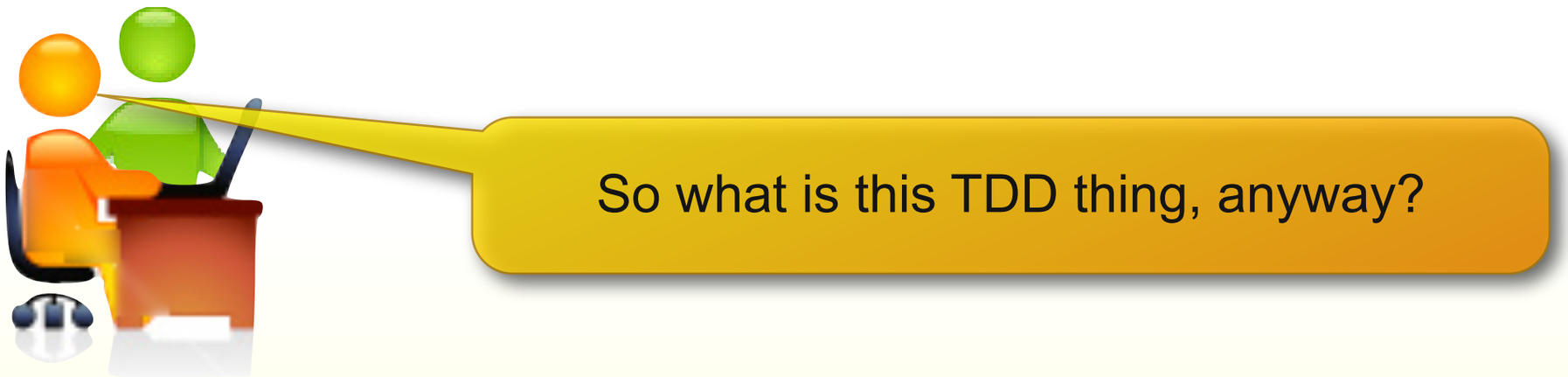
TDD Basics

TDD Basics



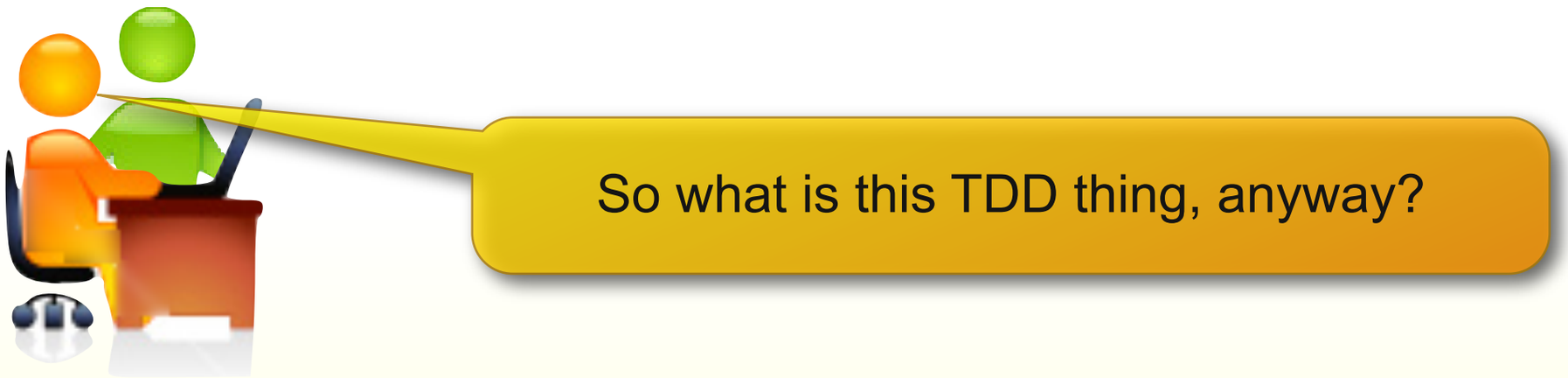
TDD Basics

- TDD is *not* about writing tests



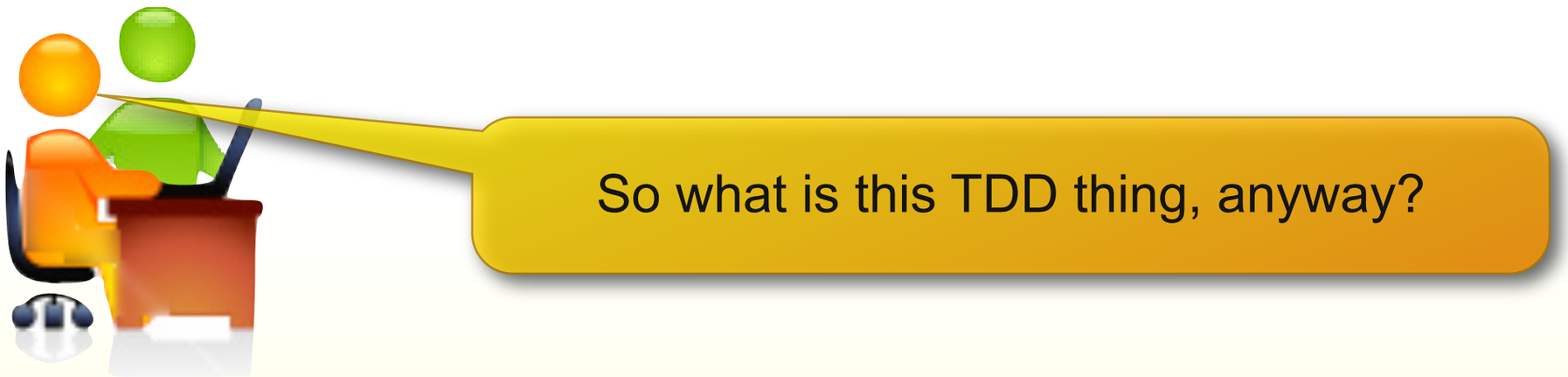
TDD Basics

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



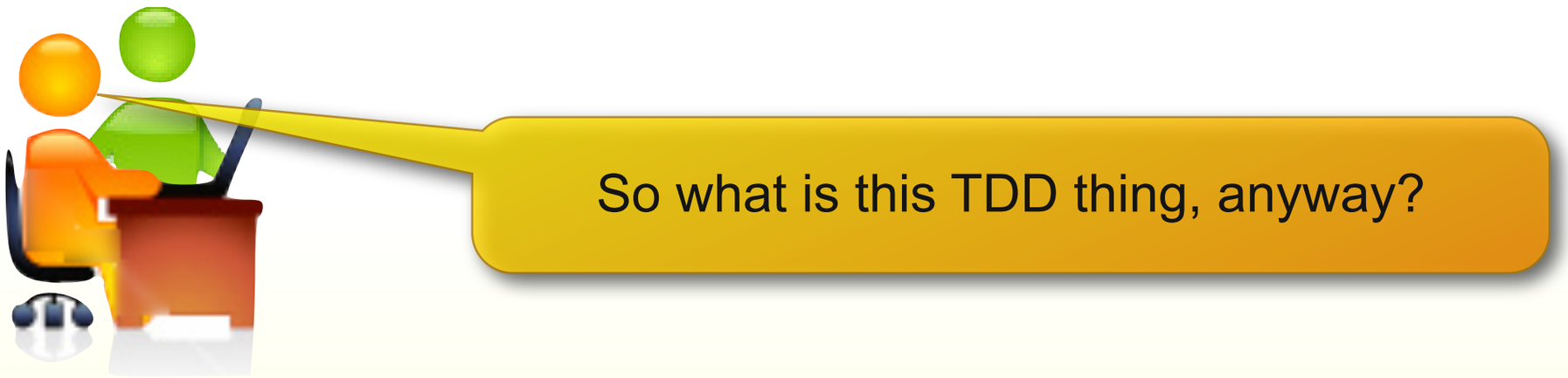
TDD Basics

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



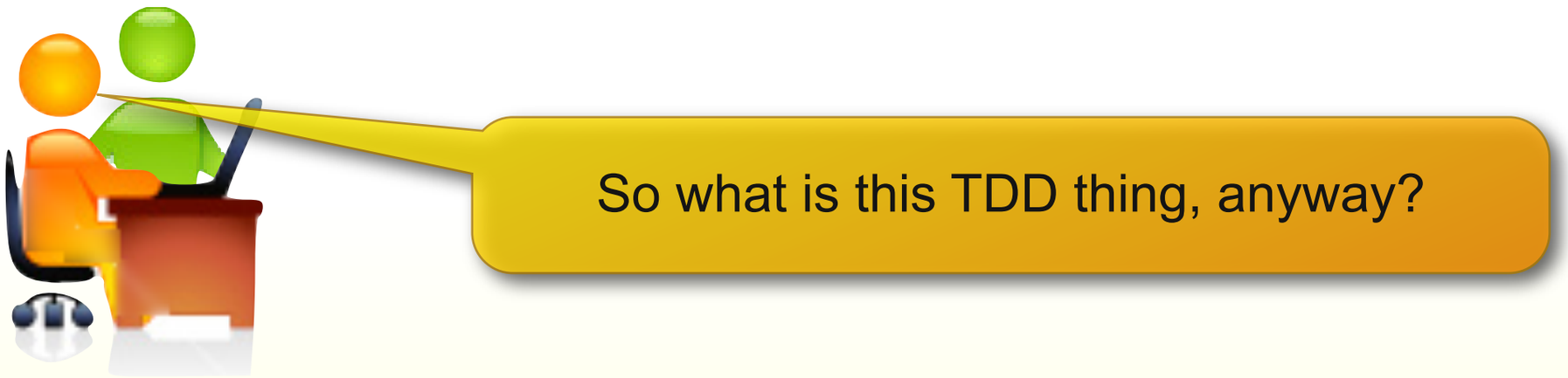
TDD Basics

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



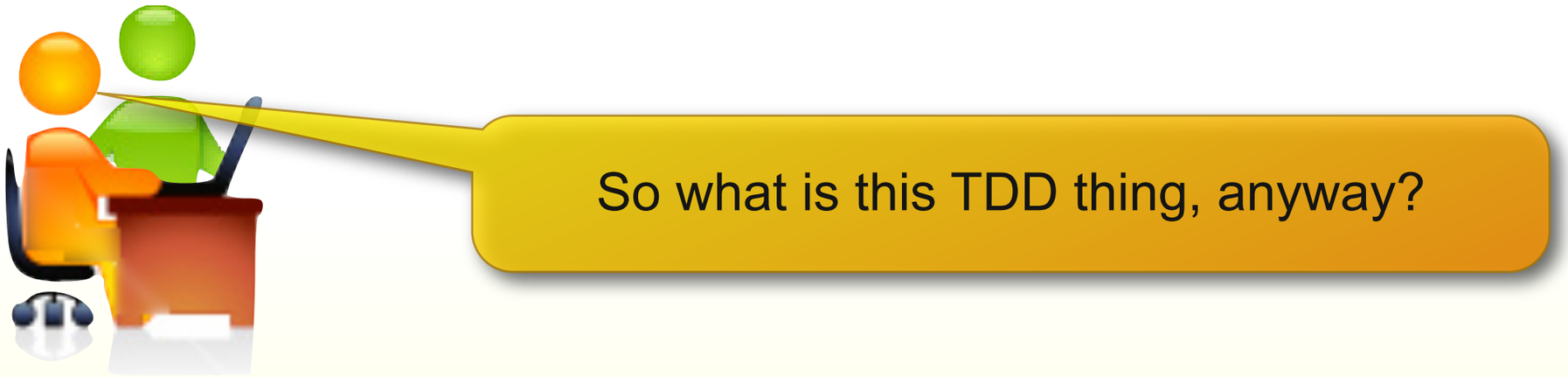
TDD Basics

- 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



TDD Basics

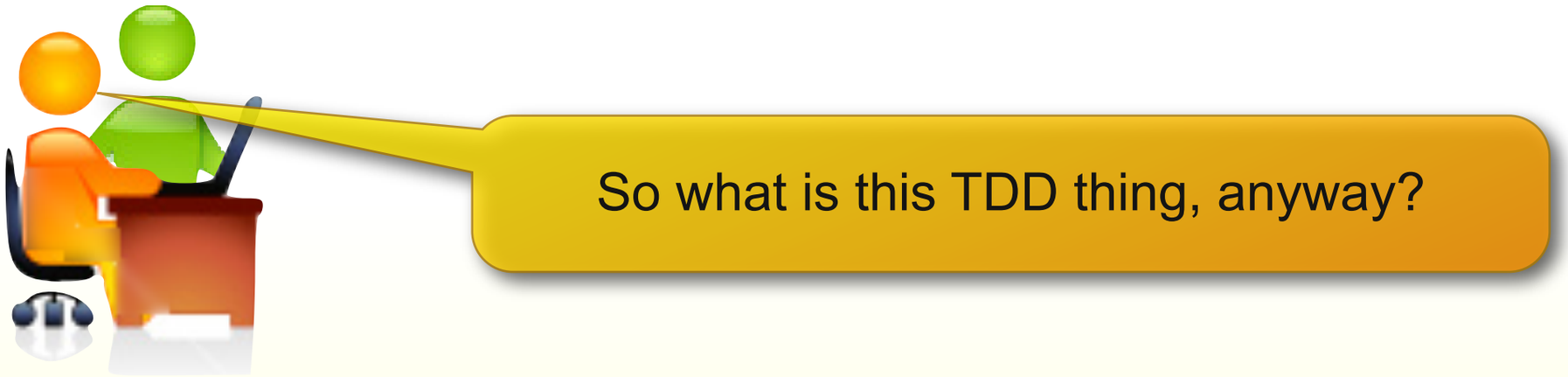
- TDD is *not* about writing tests
- TDD is a design strategy:
  - Write better-designed code
  - Have more confidence in our code
  - Make changes more easily
  - Write code that meets user requirements more accurately



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

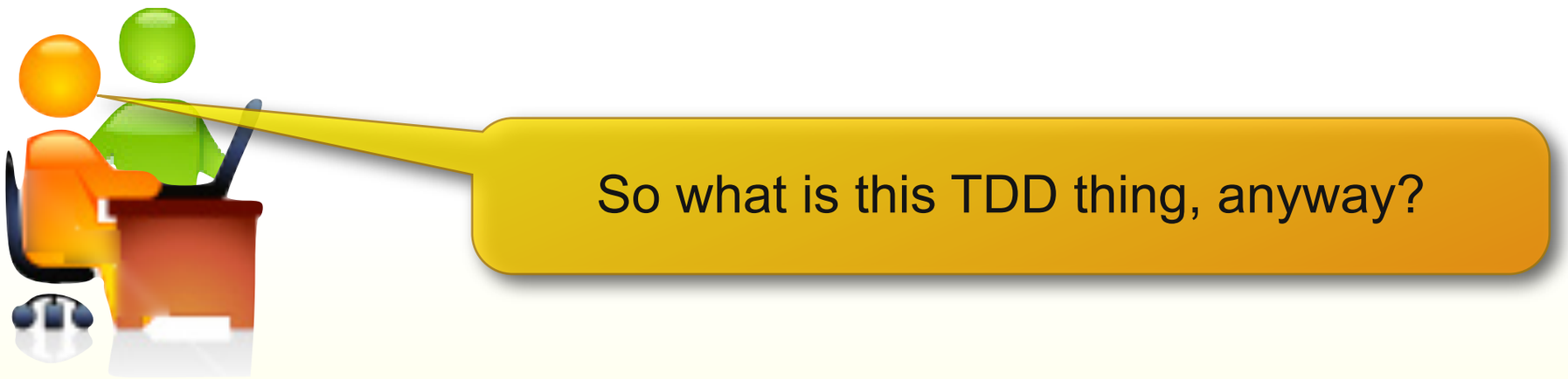
(and incidentally...)



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



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

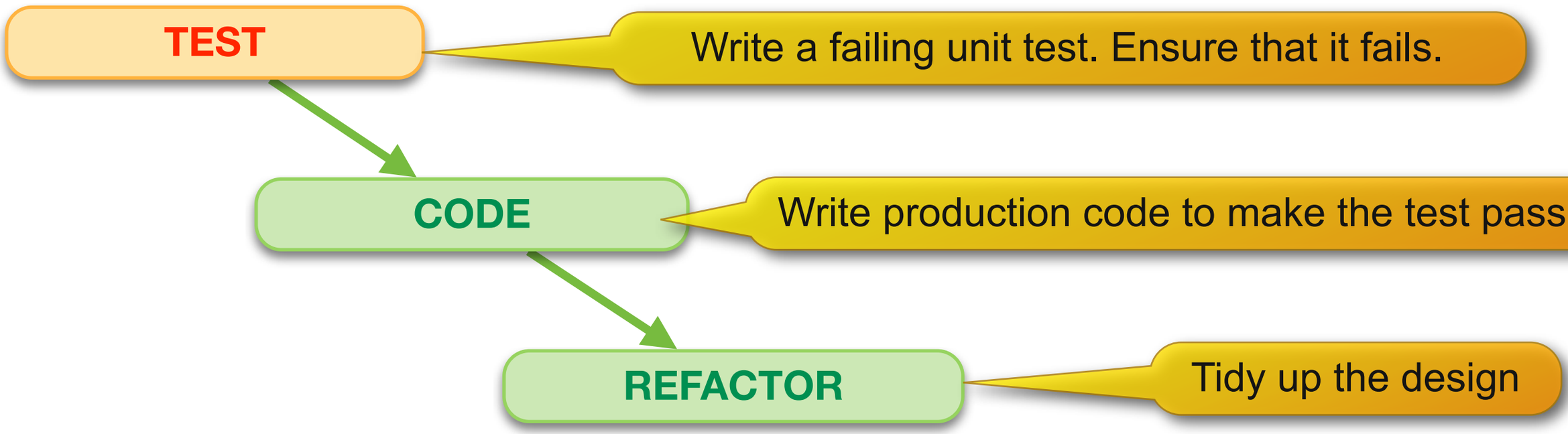
- The TDD development process

> *"Never write a single line of code unless you have a failing automated test. Eliminate duplication."*
>
> **- Kent Beck**

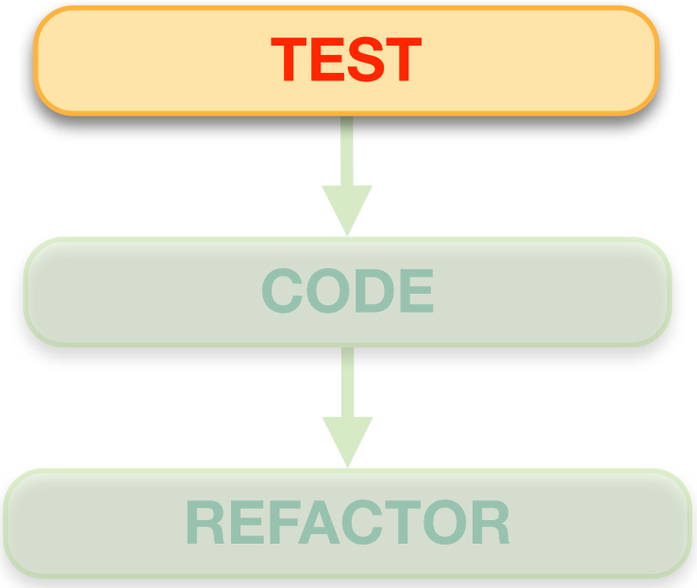
TDD Basics

- The TDD development process

> *"Never write a single line of code unless you have a failing automated test. Eliminate duplication."*
>
> **- Kent Beck**

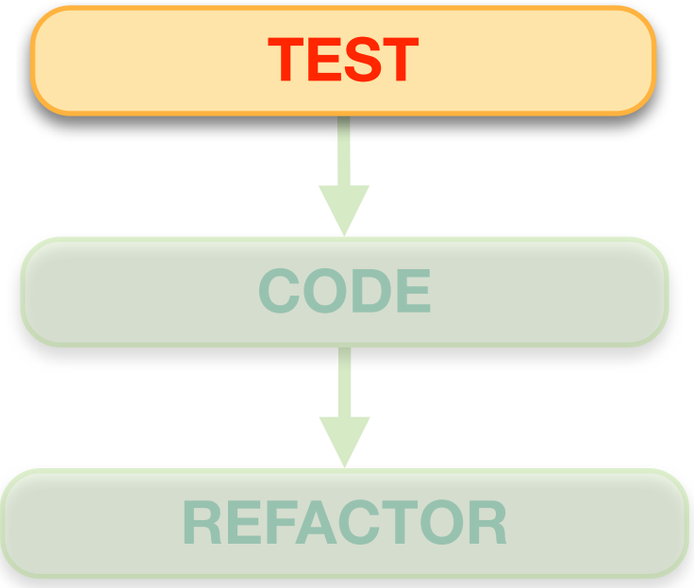


TDD Basics



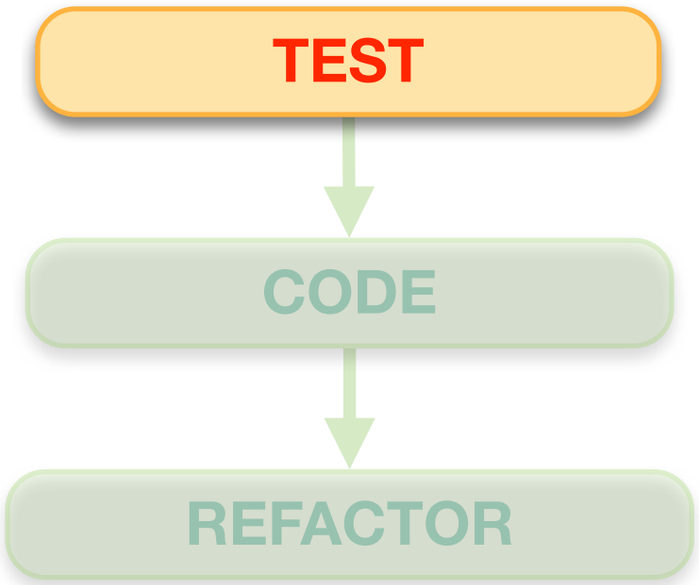
TDD Basics

- **Step 1) Write a test**



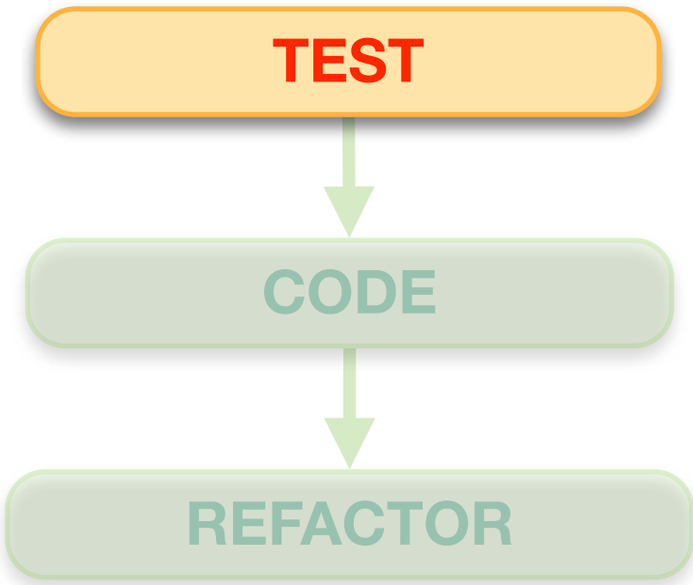
TDD Basics

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



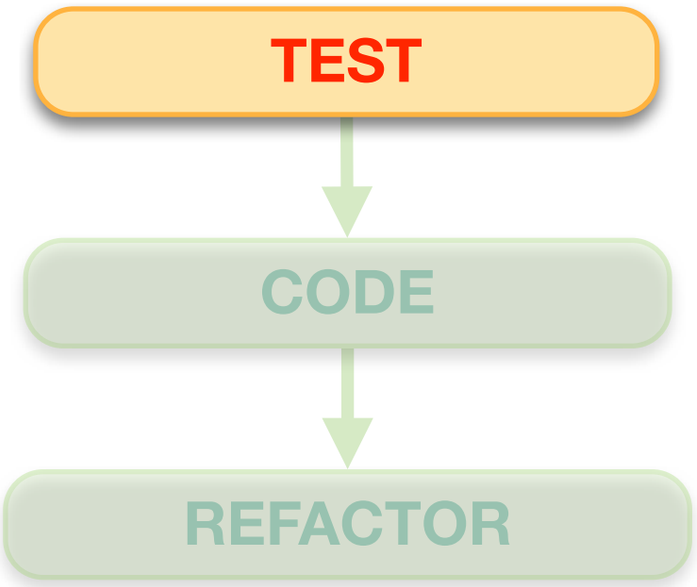
TDD Basics

- **Step 1) Write a test**
  - Design your code from the outside-in
    - How will your code be used?



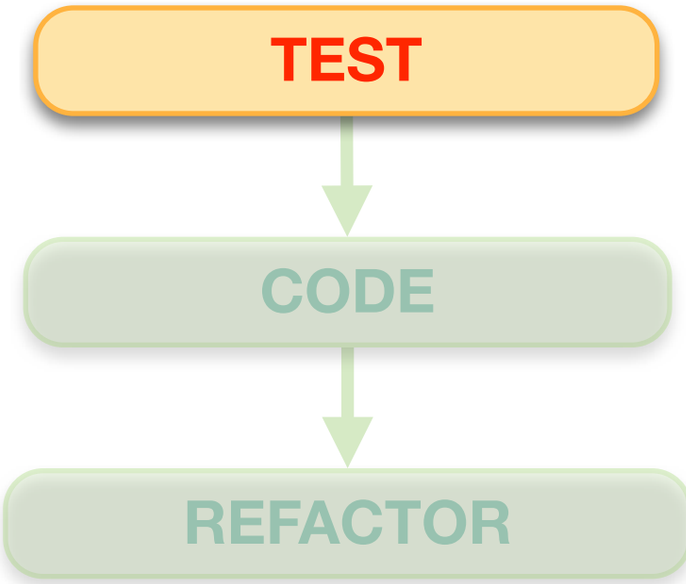
TDD Basics

- **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



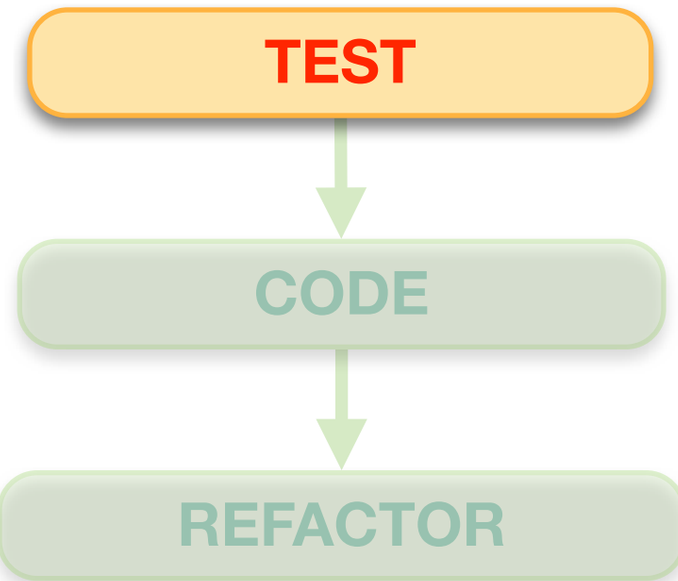
TDD Basics

- **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



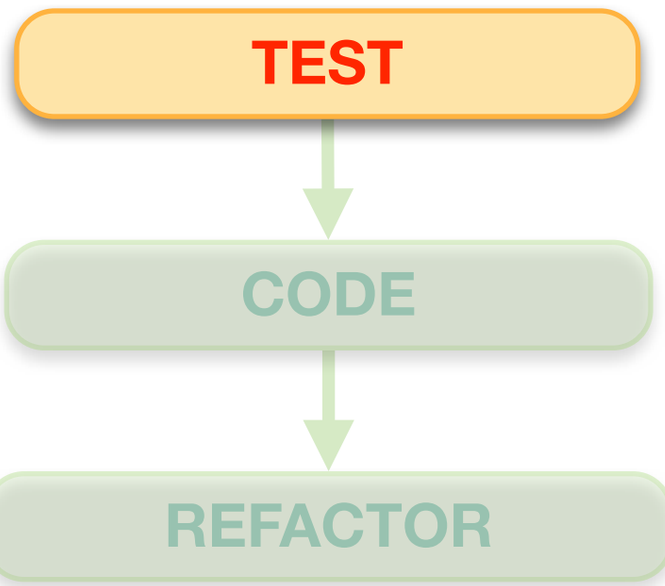
TDD Basics

- **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

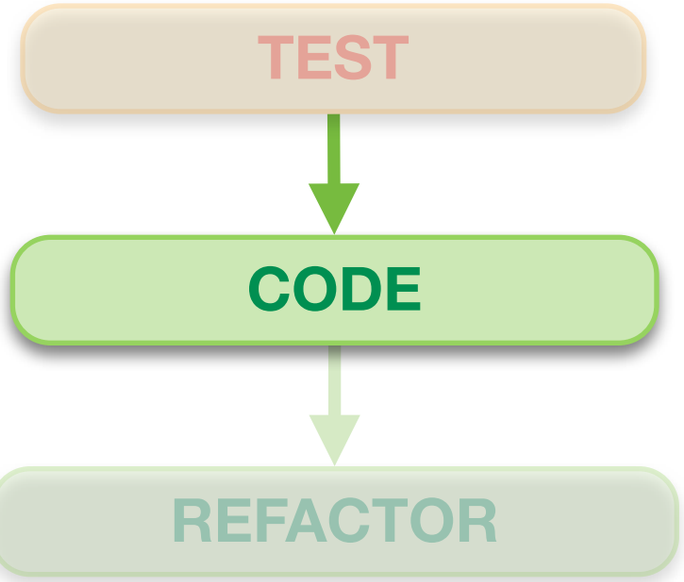


TDD Basics

- **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

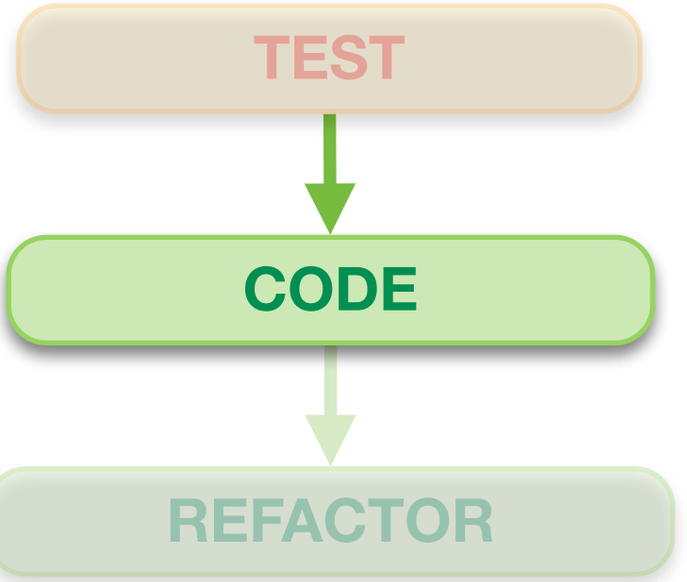


TDD Basics



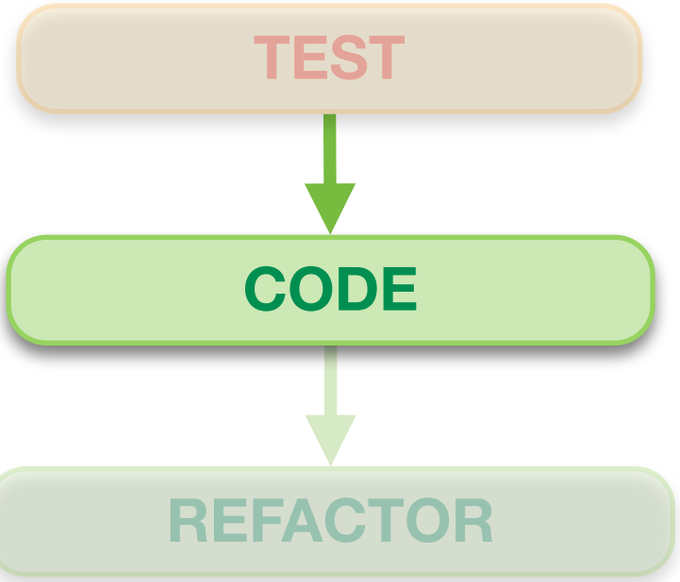
TDD Basics

- **Step 2) Write some code**



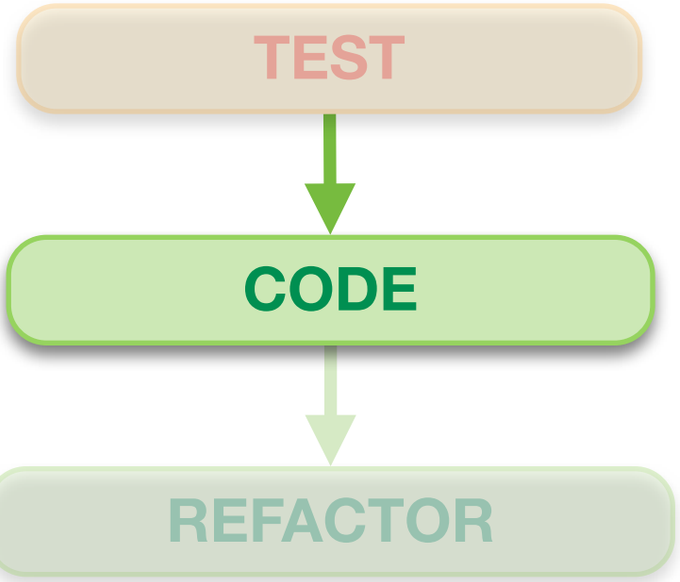
TDD Basics

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



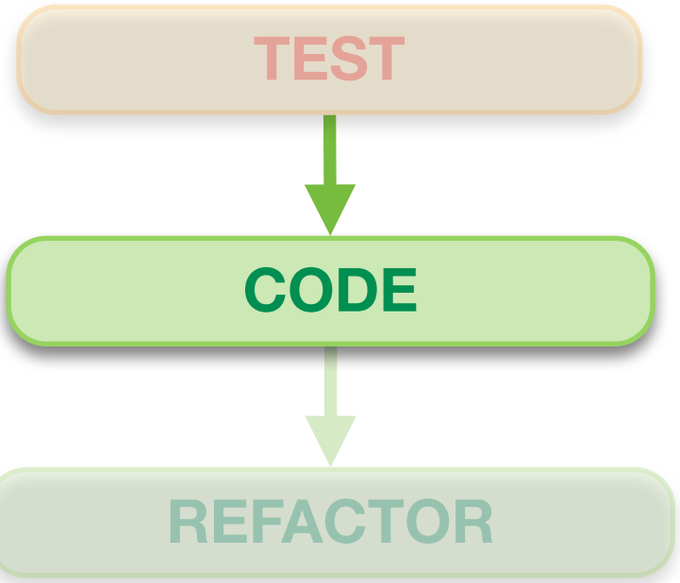
TDD Basics

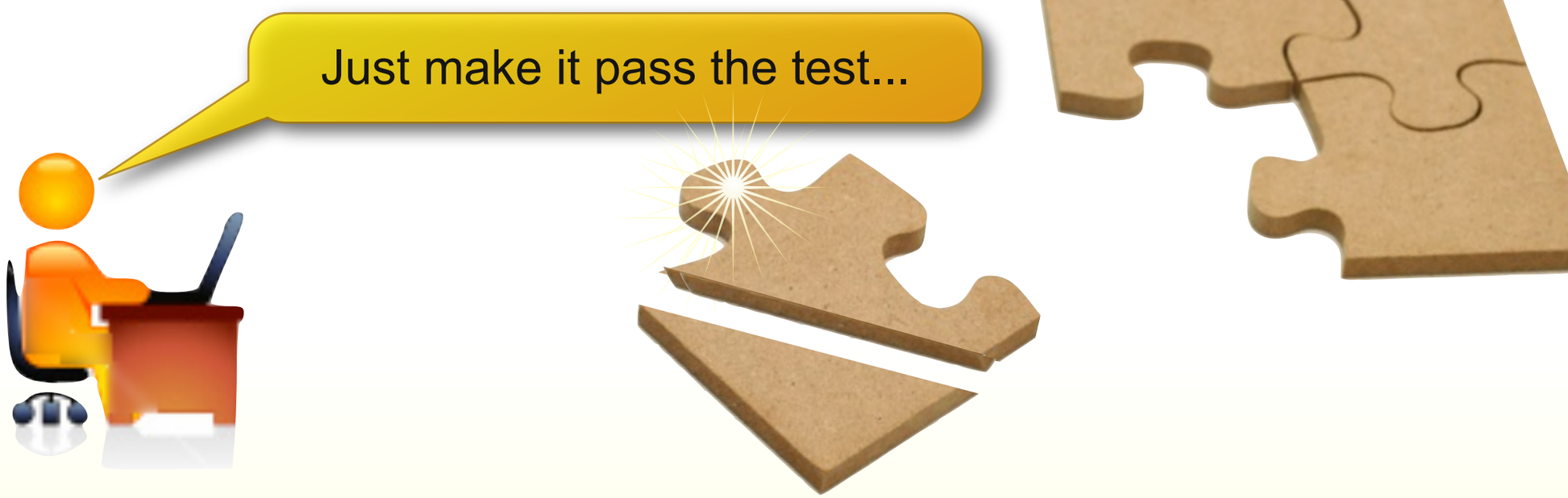
- **Step 2) Write some code**
  - Just enough code to make the tests pass
    - Might mean a non-optimal solution at this stage...



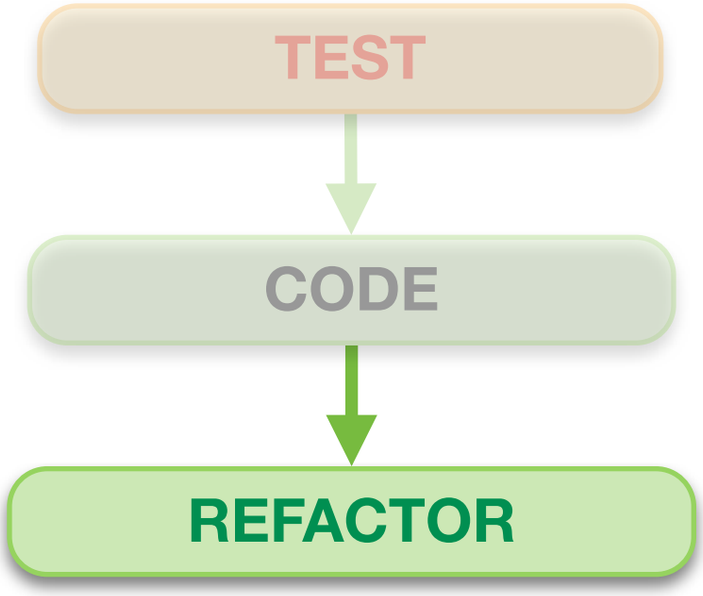
TDD Basics

- **Step 2) Write some code**
  - Just enough code to make the tests pass
    - Might mean a non-optimal solution at this stage...

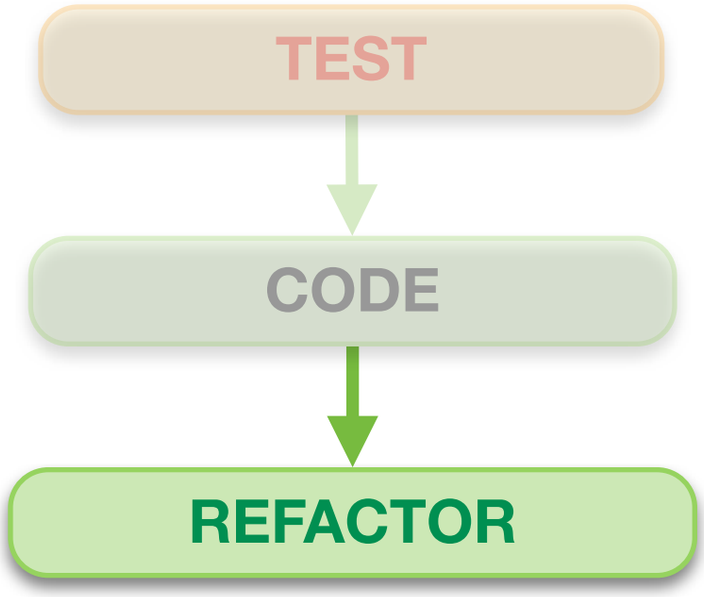


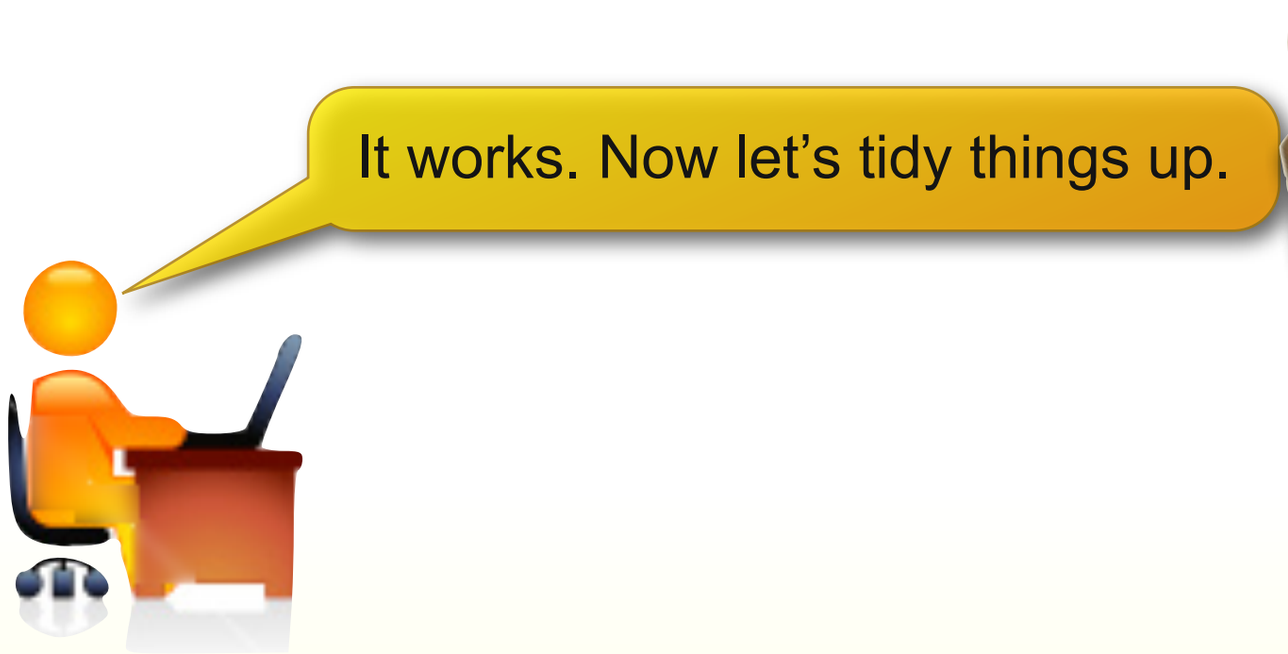


TDD Basics



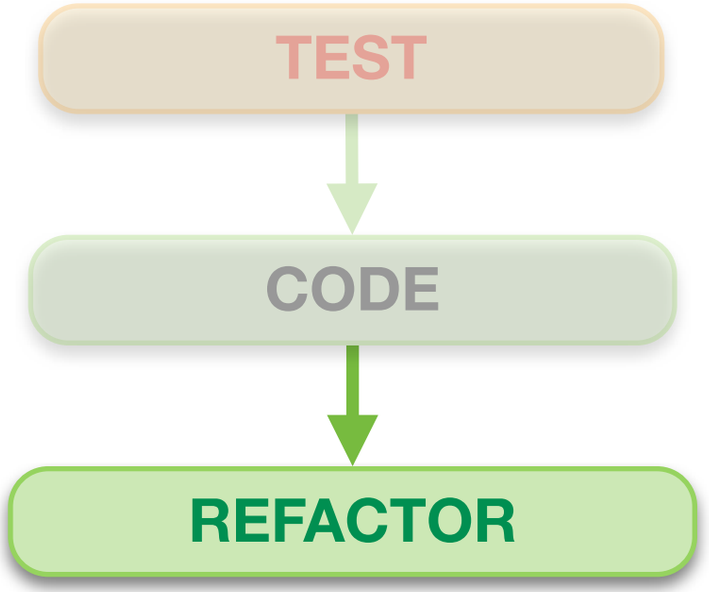
TDD Basics

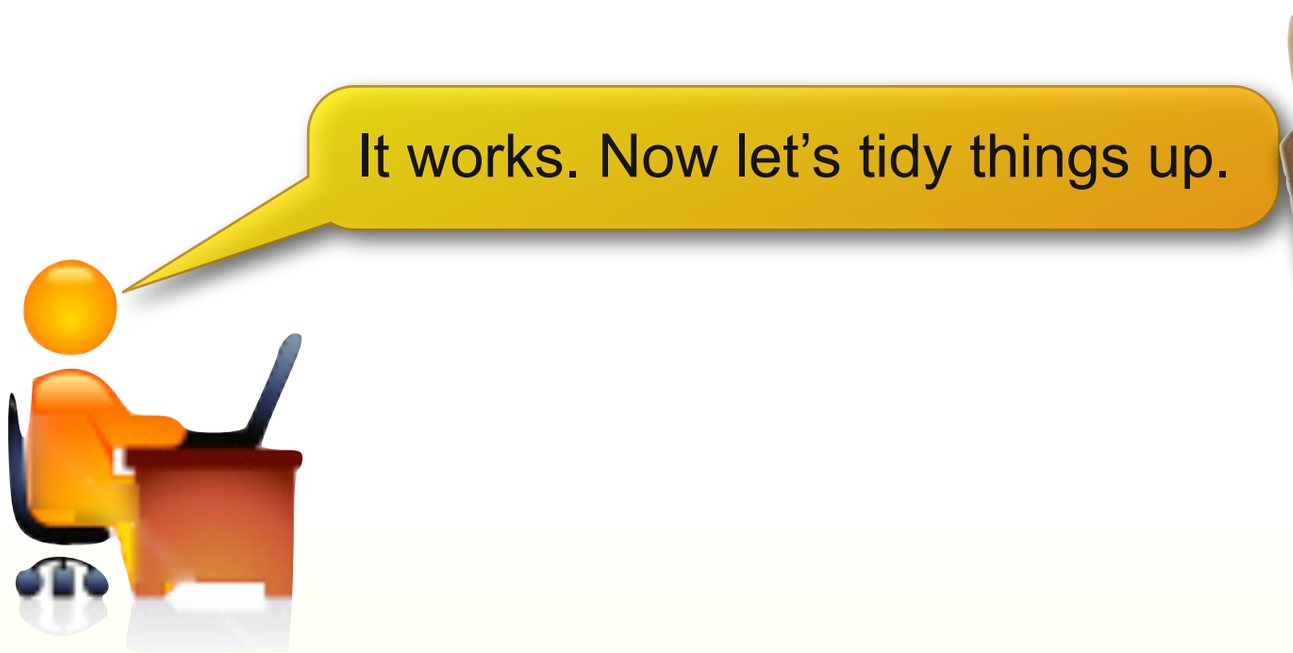




TDD Basics

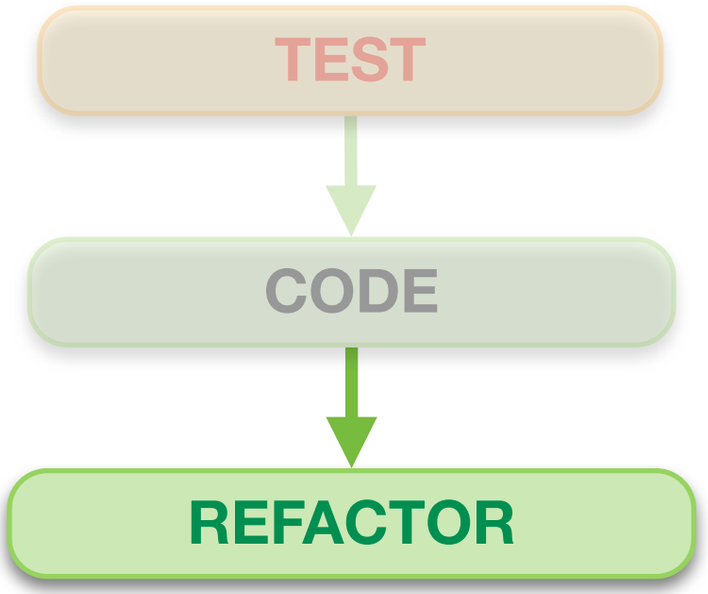
- **Step 3) Tidy up**

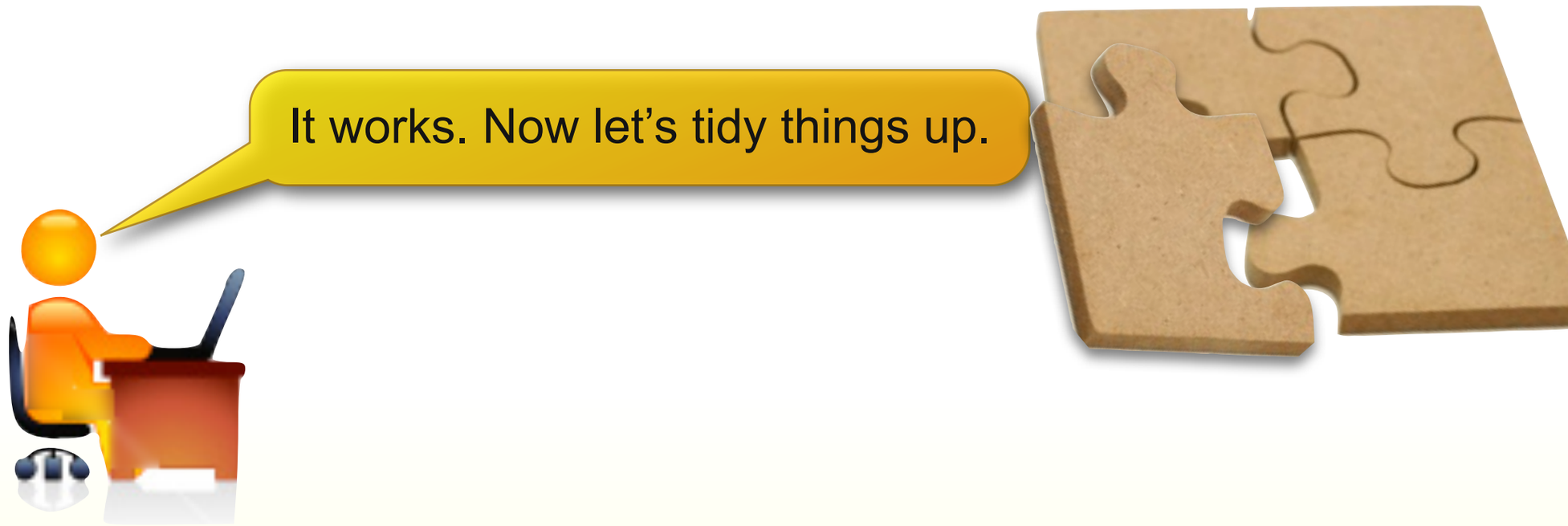




TDD Basics

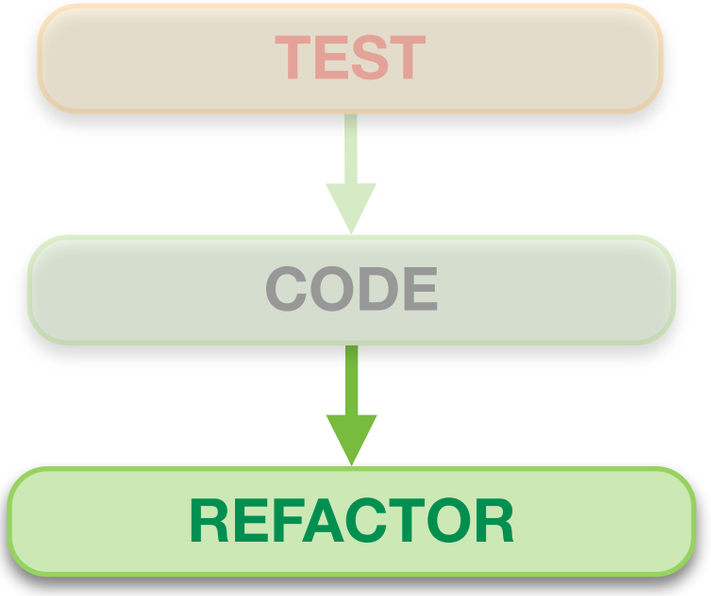
- **Step 3) Tidy up**
  - Evolutionary (emerging) design

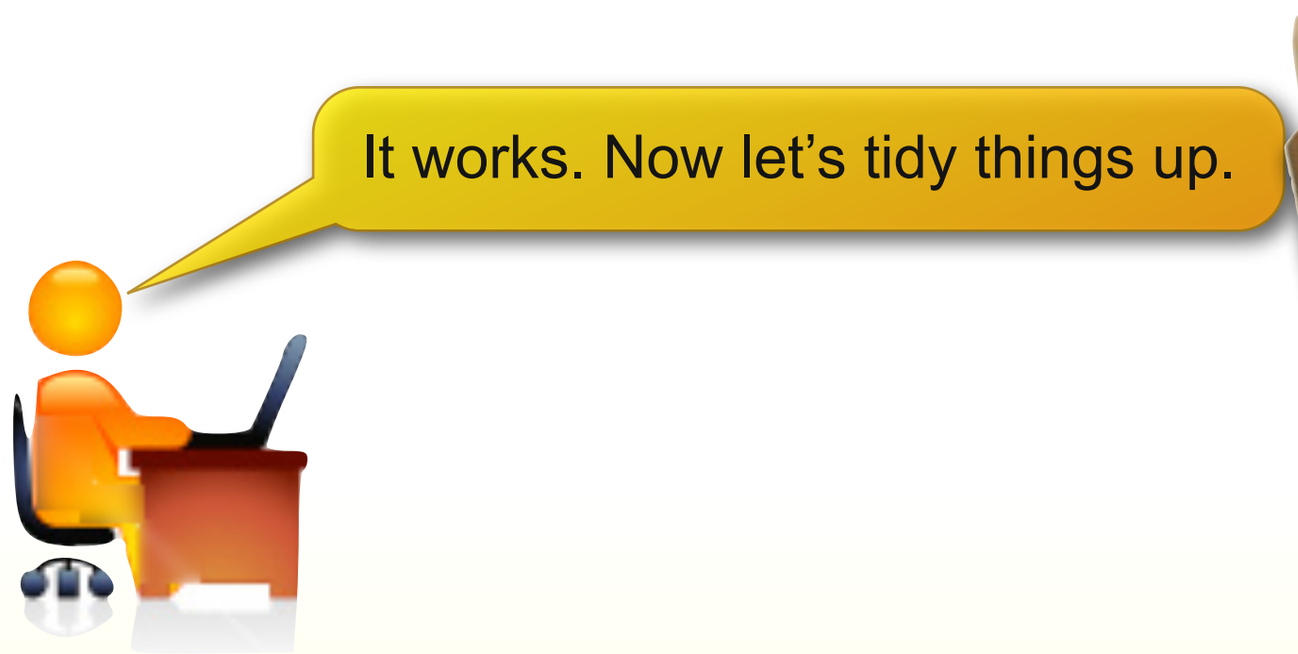




TDD Basics

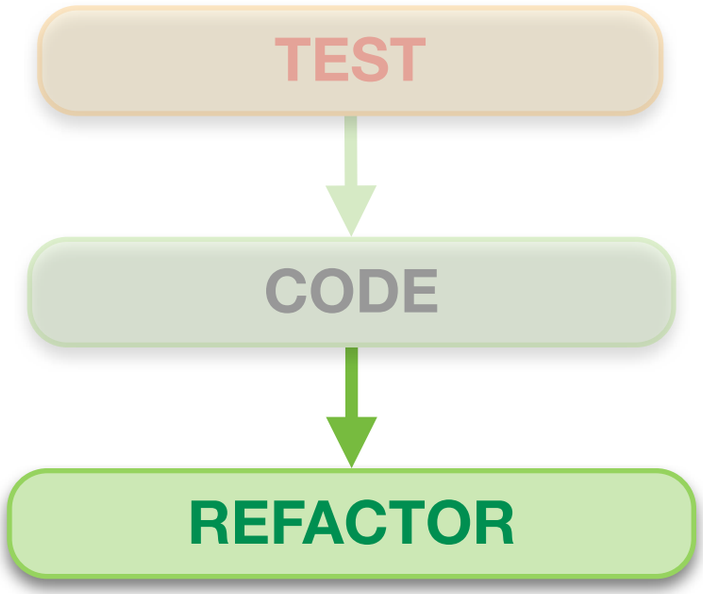
- **Step 3) Tidy up**
  - Evolutionary (emerging) design
    - Keep the code clean!

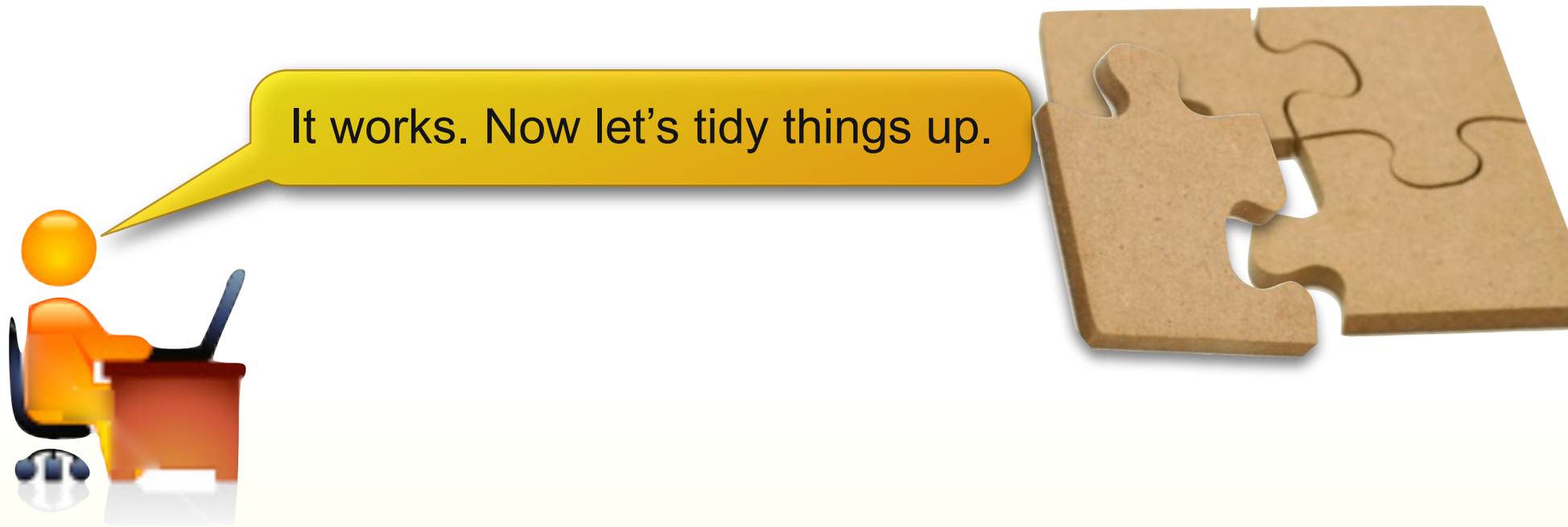




TDD Basics

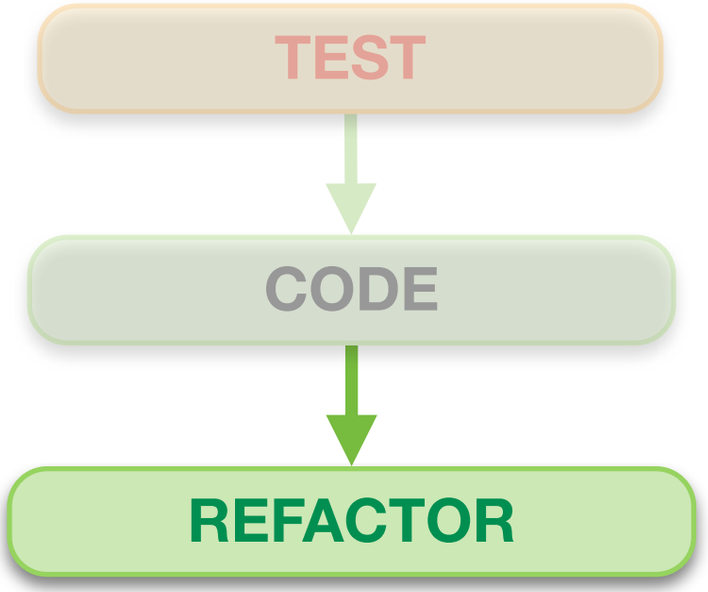
- **Step 3) Tidy up**
  - Evolutionary (emerging) design
    - Keep the code clean!
    - Remove duplication and fix design problems

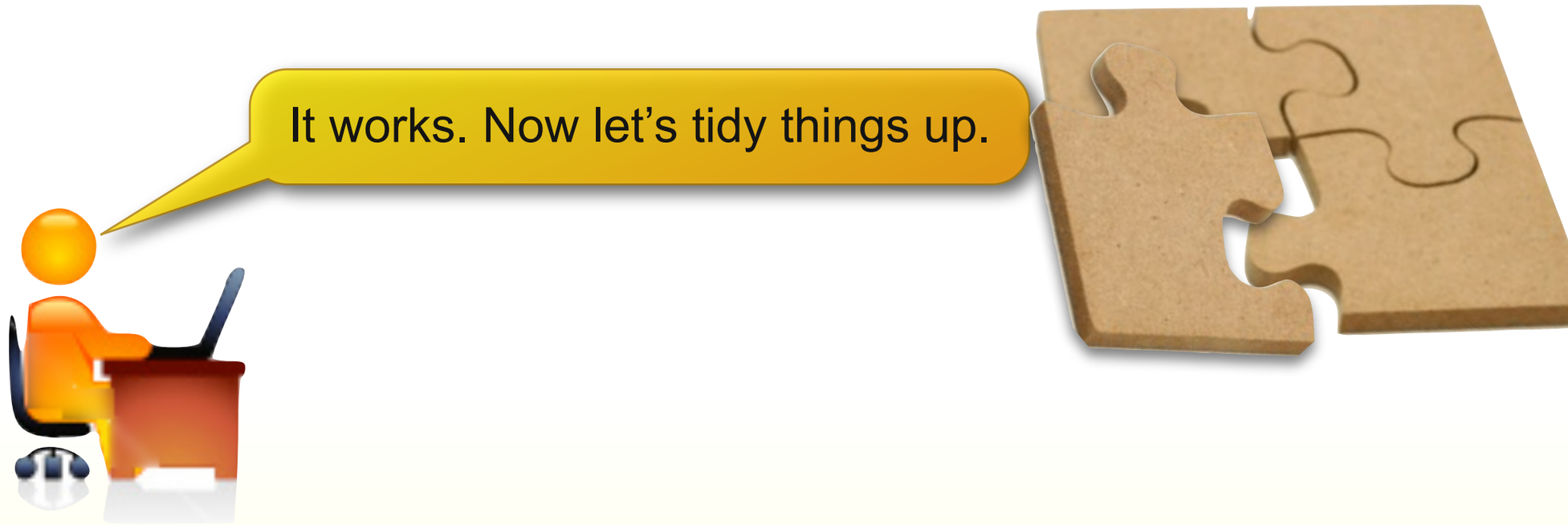




TDD Basics

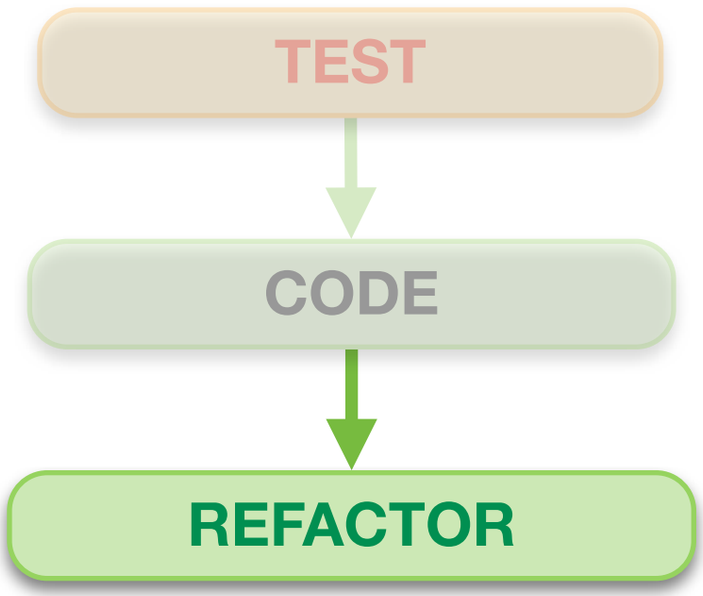
- **Step 3) Tidy up**
  - Evolutionary (emerging) design
    - Keep the code clean!
    - Remove duplication and fix design problems
    - Restructure existing code - no new features!

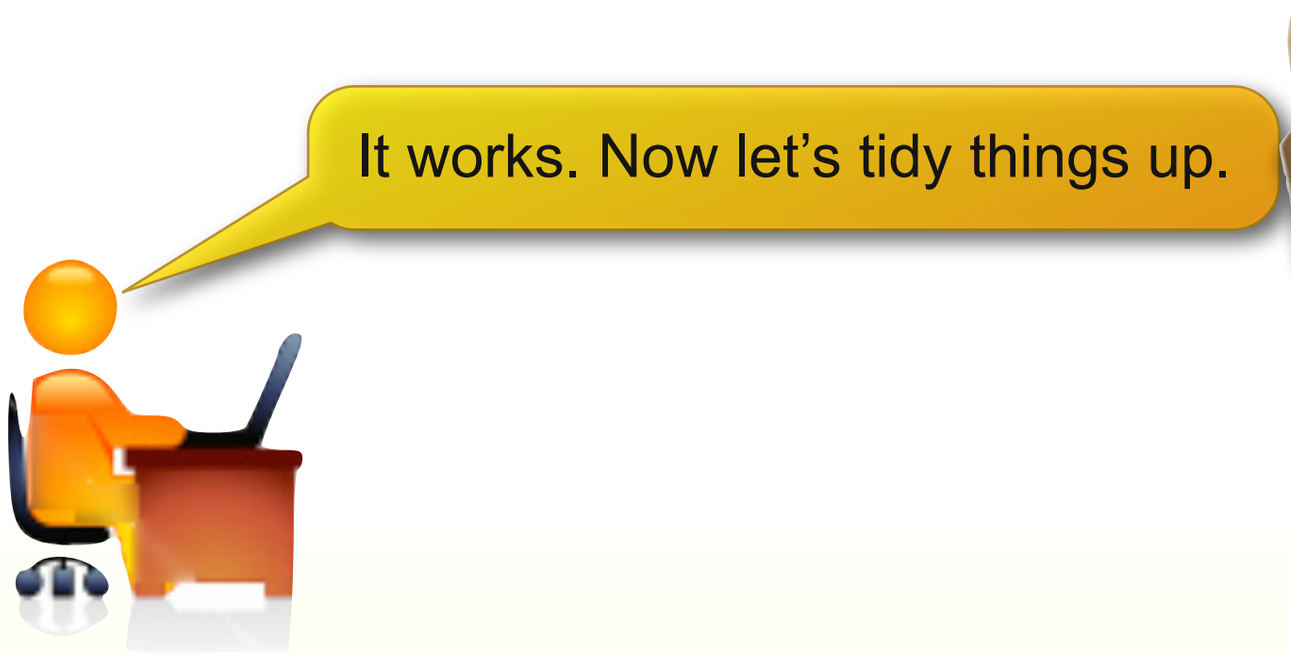




TDD Basics

- **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

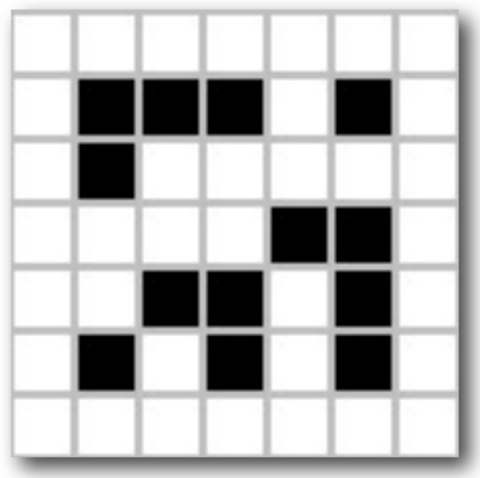
TDD in Action - the Game Of Life

TDD in Action - the Game Of Life

TDD in Action - the Game Of Life

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



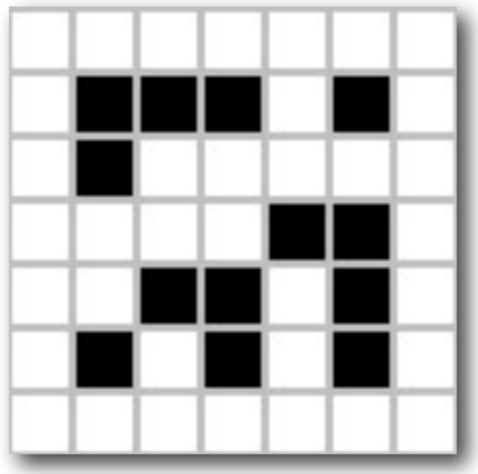
TDD in Action

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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

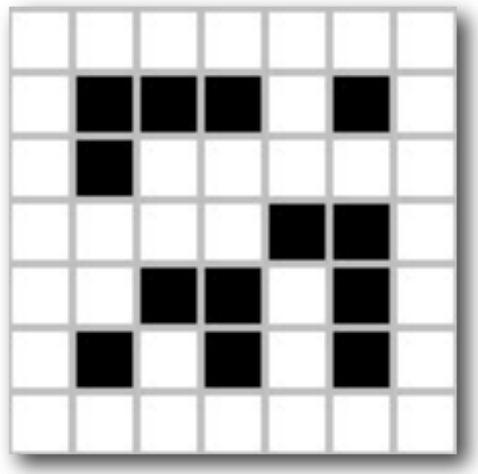
Now what?

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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?

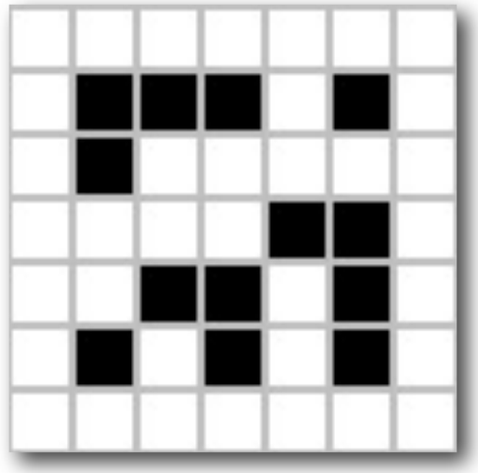
So what do we mean by this first requirement?

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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



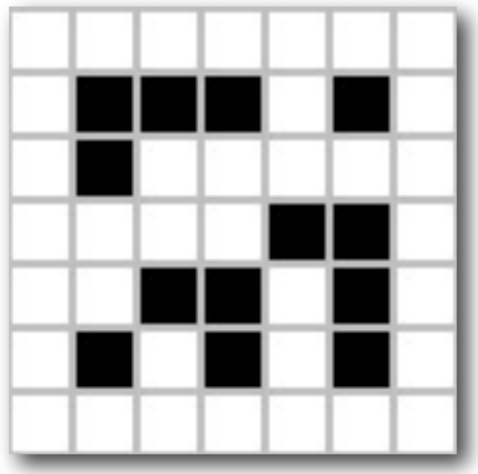
TDD in Action

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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

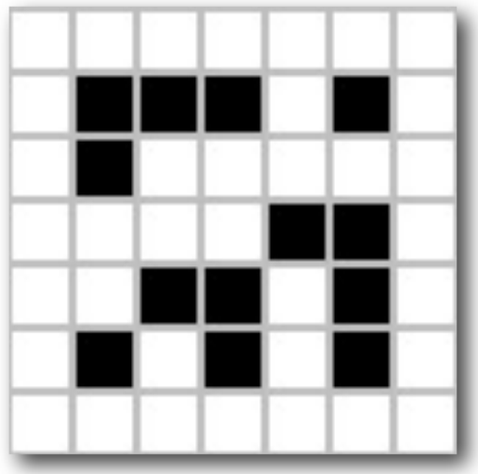
So what do we have to code for this one.

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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.

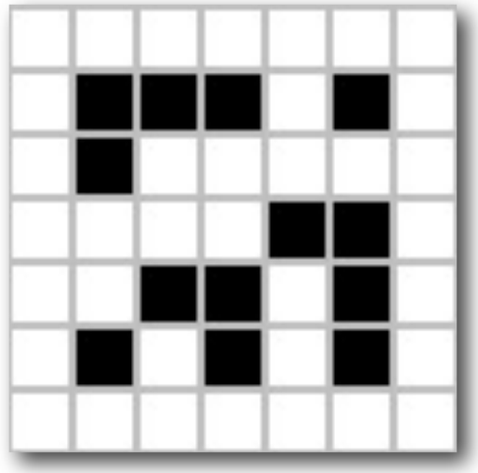
Dunno yet. Let's write some acceptance tests to find out

The Game of Life

- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

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.

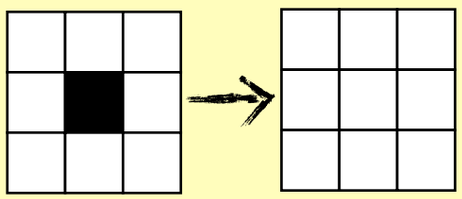
Dunno yet. Let's write some acceptance tests to find out

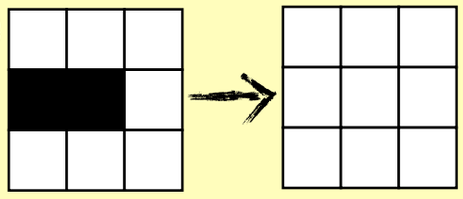
The Game of Life

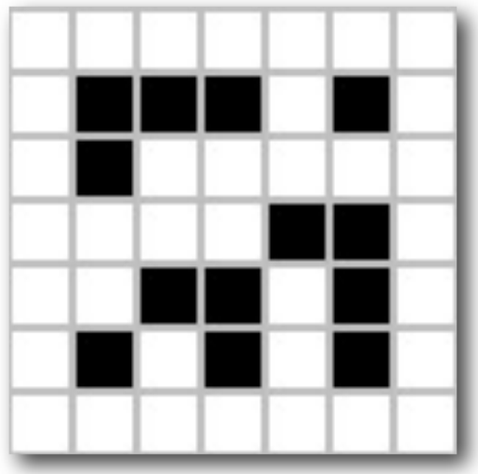
- Live cells with less than 2 live neighbours die
- [illegible]
- [illegible]
- [illegible]

User Story 1 - Acceptance criteria

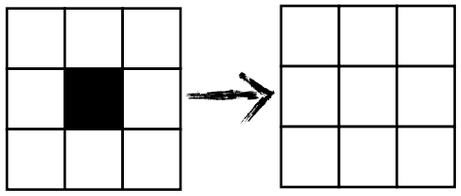
Sample grid transitions:

1) 

2) 

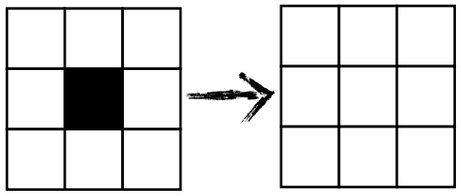


TDD in Action



TDD in Action

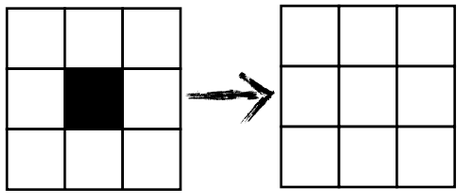
Great, so now I can start coding, right?



TDD in Action

Great, so now I can start coding, right?

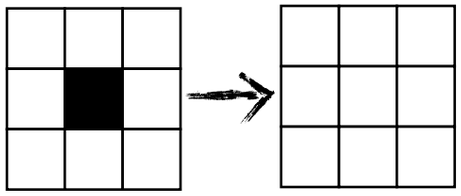
Not so fast, hot shot. We need to turn this into an ‘executable requirement’



TDD in Action

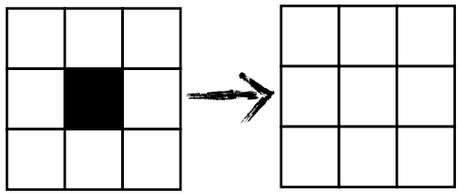
Great, so now I can start coding, right?

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



```
public class GameOfLifeTest {

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

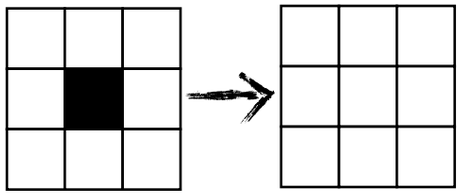
TDD in Action



```
public class GameOfLifeTest {

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

TDD in Action

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

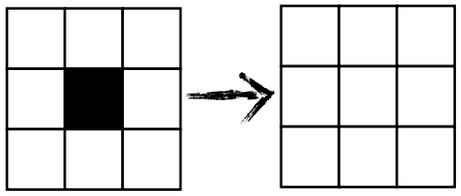


```
public class GameOfLifeTest {

    @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

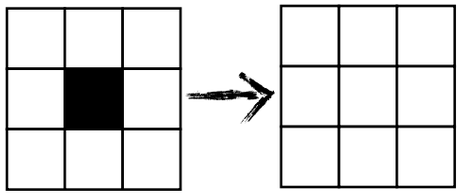


```
public class GameOfLifeTest {

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

TDD in Action

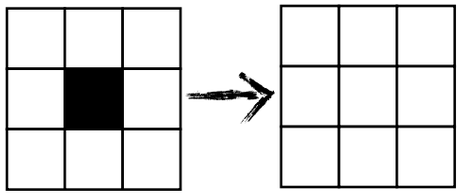
I still don’t know what to write in this test...

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));
    }
}
```

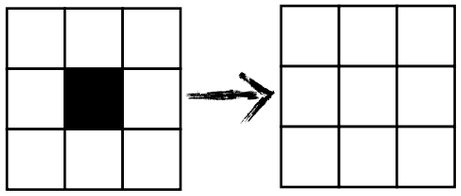
TDD in Action



```
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?

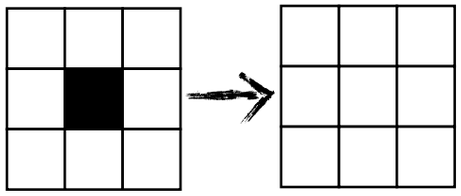


```
public class GameOfLifeTest {

    @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?

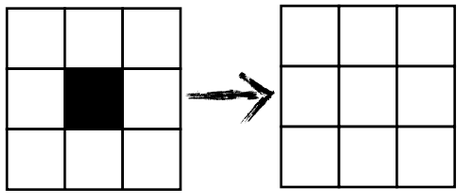


```
public class GameOfLifeTest {

    @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

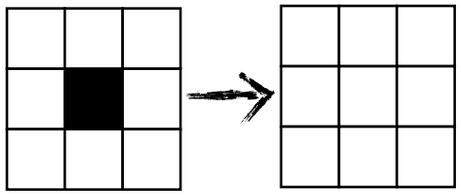
```
public class GameOfLifeTest {

    @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

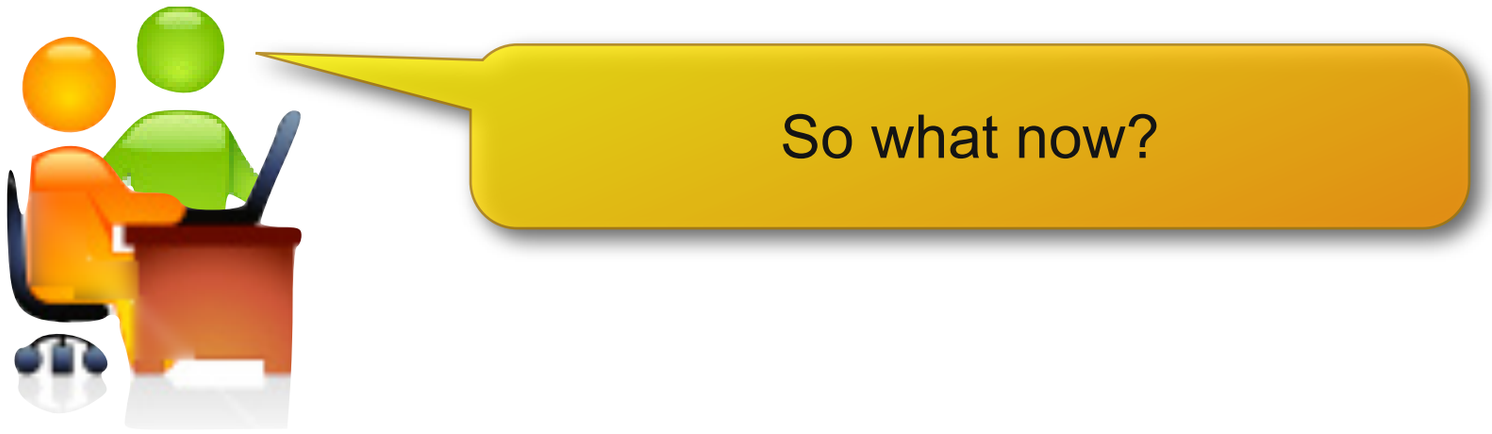
We started by expressing our expectations

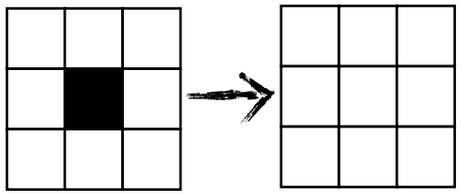
TDD in Action



```
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



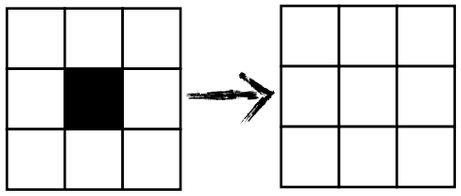


```
public class GameOfLifeTest {

    @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.

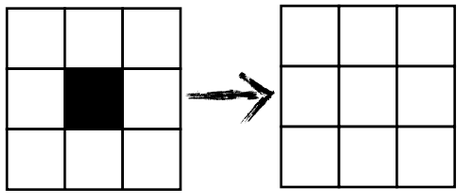


```
public class GameOfLifeTest {

    @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

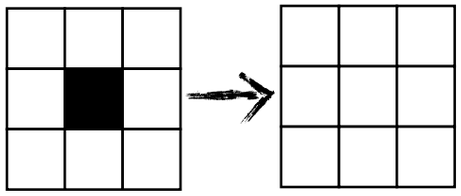
So what now?

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));
    }
}
```

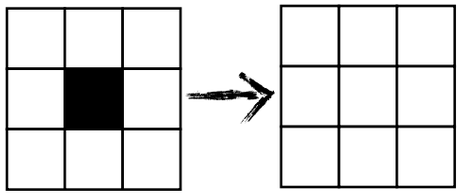
TDD in Action



```
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?

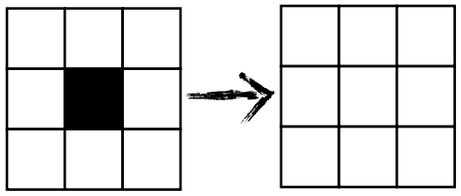


```
public class GameOfLifeTest {

    @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!

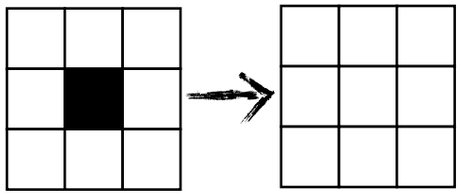


```
public class GameOfLifeTest {

    @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?

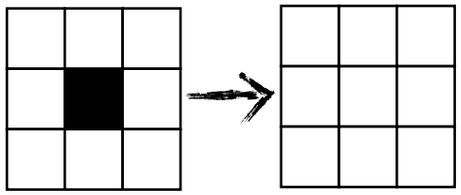
Spot on! You catch on fast!



```
public class UniverseTest {

    @Test
    public void aUniverseSeededWithAnEmpyGridContentWillContainAnEmptyGrid() {

        String seededGrid = "...\n" +
                            "...\n" +
                            "...";

        Universe theUniverse = new Universe(seededWith(seededGrid));
        String currentGrid = theUniverse.getGrid();
        assertThat(currentGrid, is(seededGrid));
    }
}
```

```
public

    @T
    pu

        Universe theUniverse = new Universe(seededWith(initialGrid));
        theUniverse.createNextGeneration();

        String nextGrid = theUniverse.getGrid();
        assertThat(nextGrid, is(expectedNextGrid));
    }
}
```

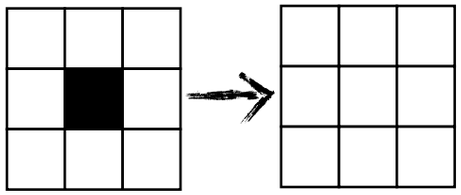
TDD in Action



```
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));
    }
}
```

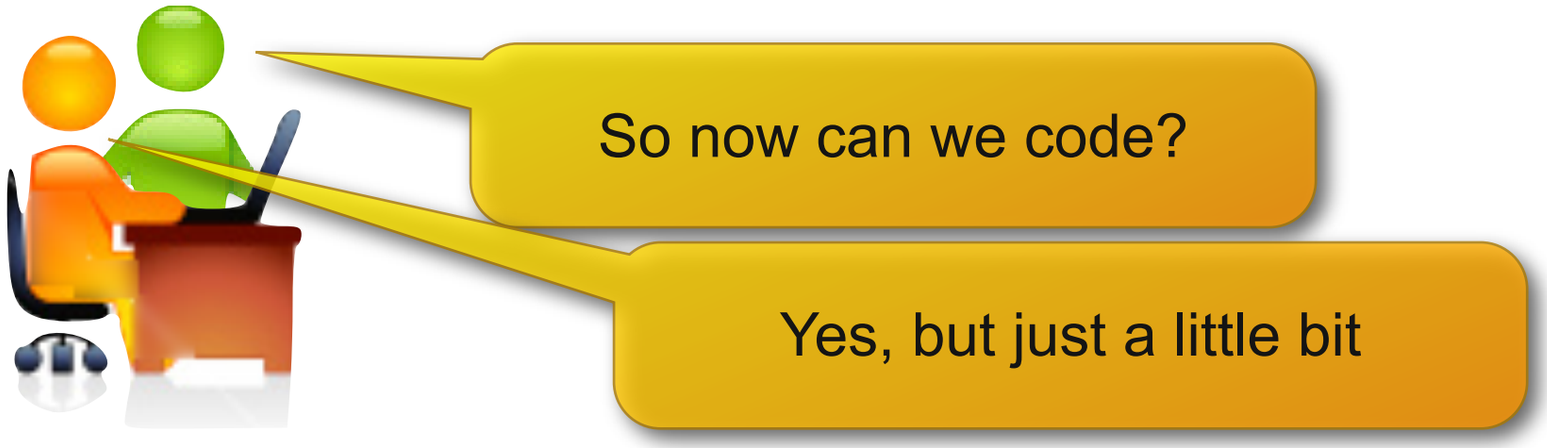
TDD in Action

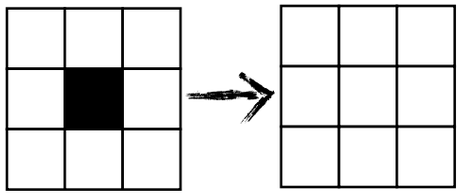
So now can we code?



```
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



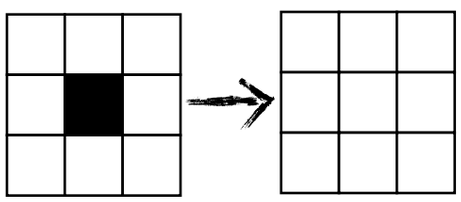


```
public class UniverseTest {

    @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?

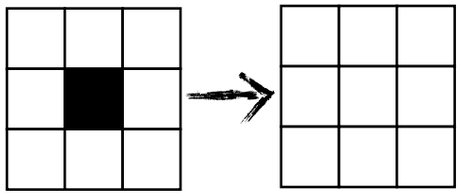
Yes, but just a little bit



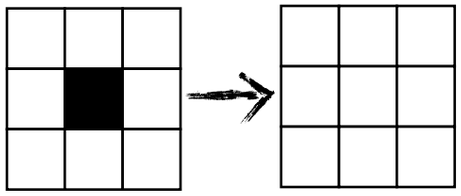
```
public class UniverseTest {

    @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" +
               "...";
    }
}
```

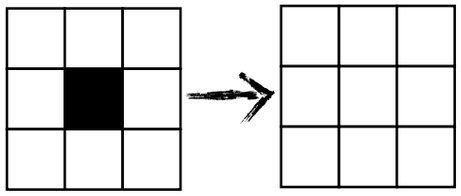
TDD in Action



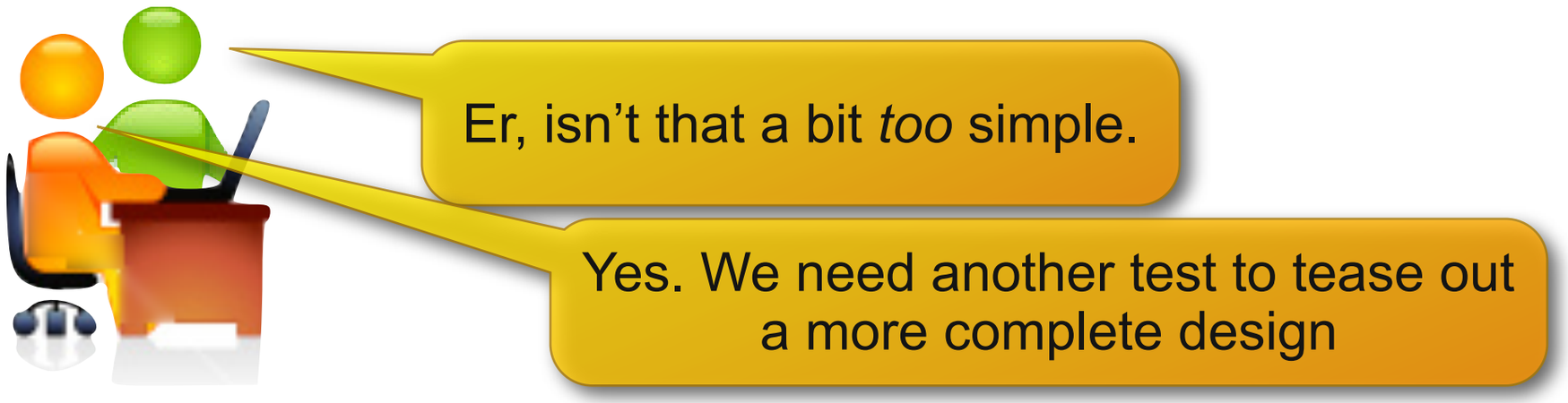
TDD in Action

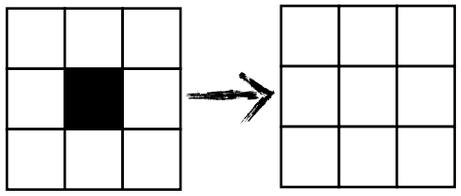


TDD in Action



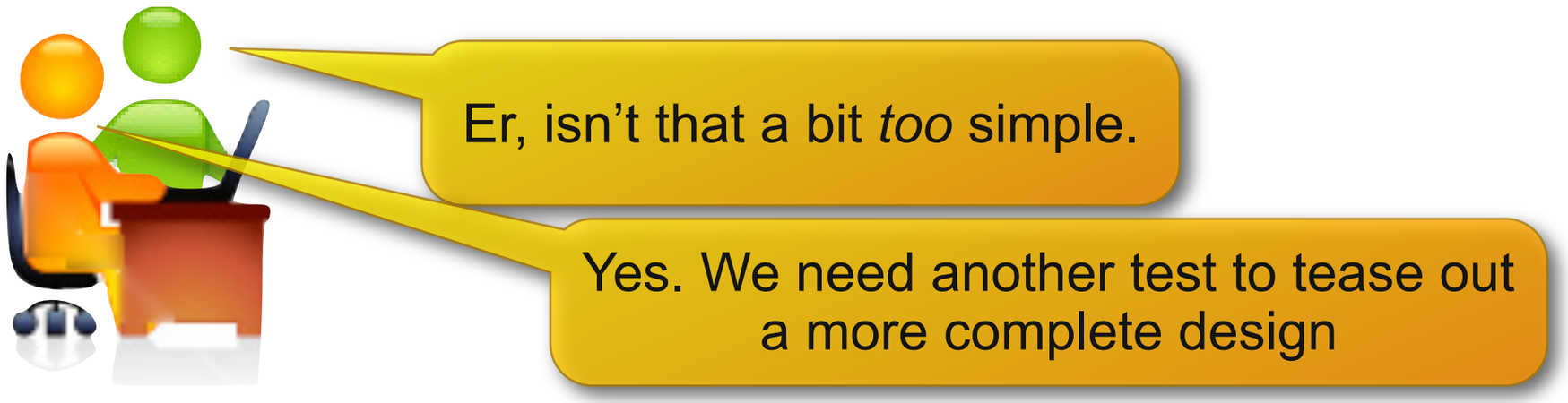
TDD in Action

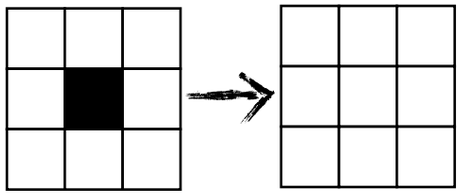




```
public class UniverseTest {

    @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



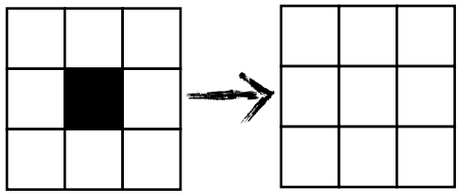


```
public class UniverseTest {

    @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.

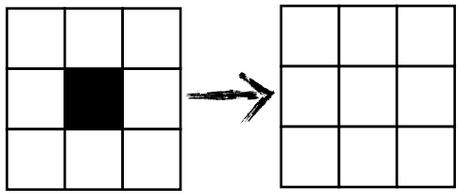
Yes. We need another test to tease out a more complete design



```
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;
    }
}
```

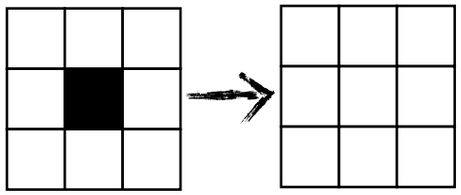
TDD in Action



```
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.

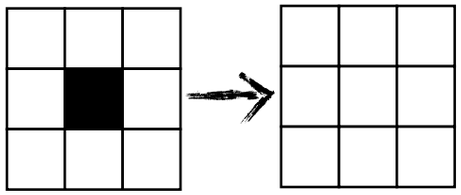


```
public class Universe {

    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

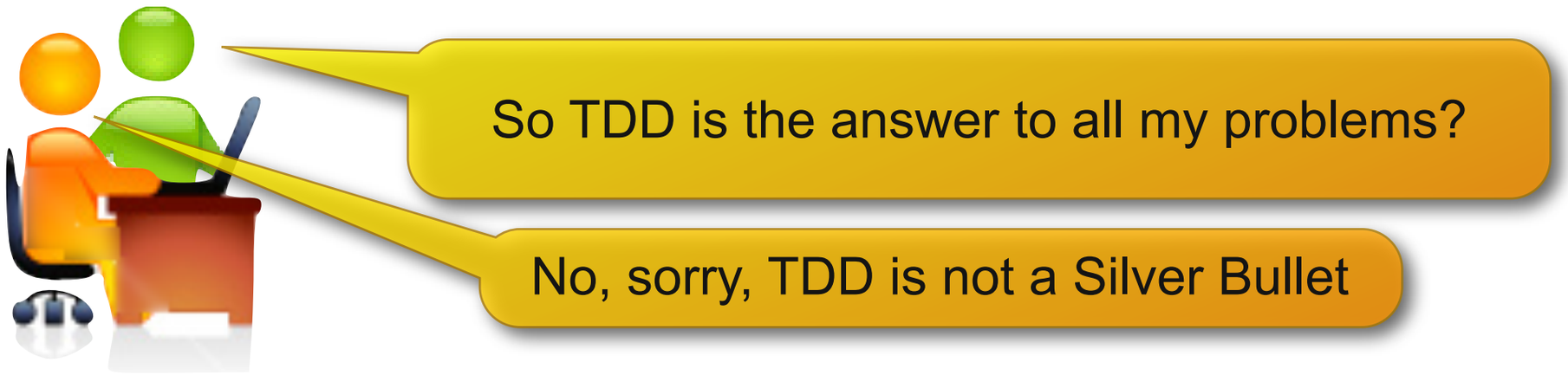
Well, now at least those two tests pass.

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



- 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

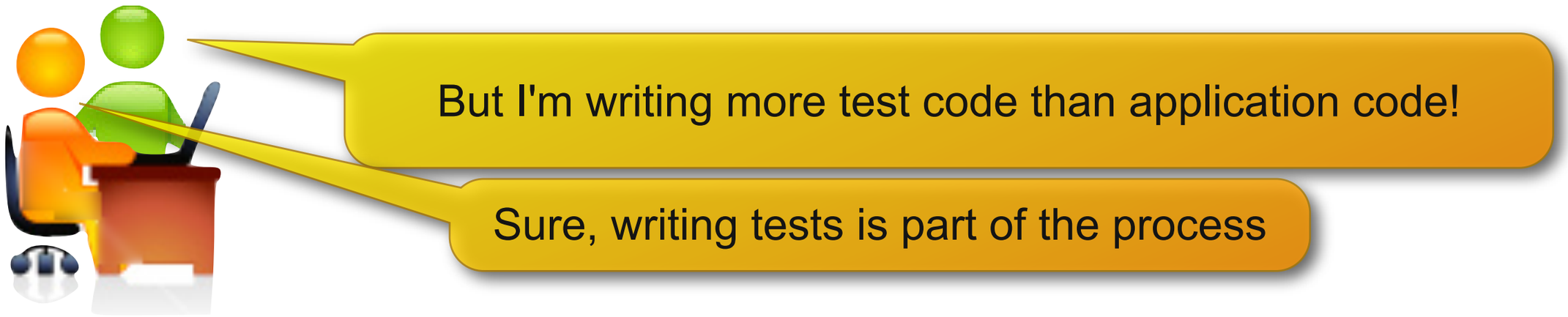
But you have time to fix the code afterwards, right?

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

- Kent Beck (paraphrased)

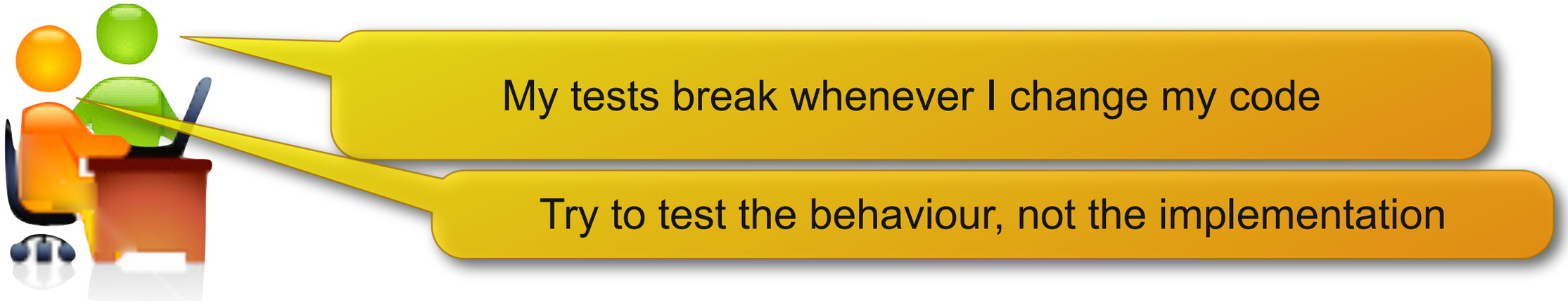
- New TDDers will take maybe 20-30% longer
- Experienced TDD developers don’t spend much (if any) more time coding
- But the code is of much higher quality

TDD for unbelievers



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

TDD for unbelievers



- Validate behaviour, don’t verify implementation

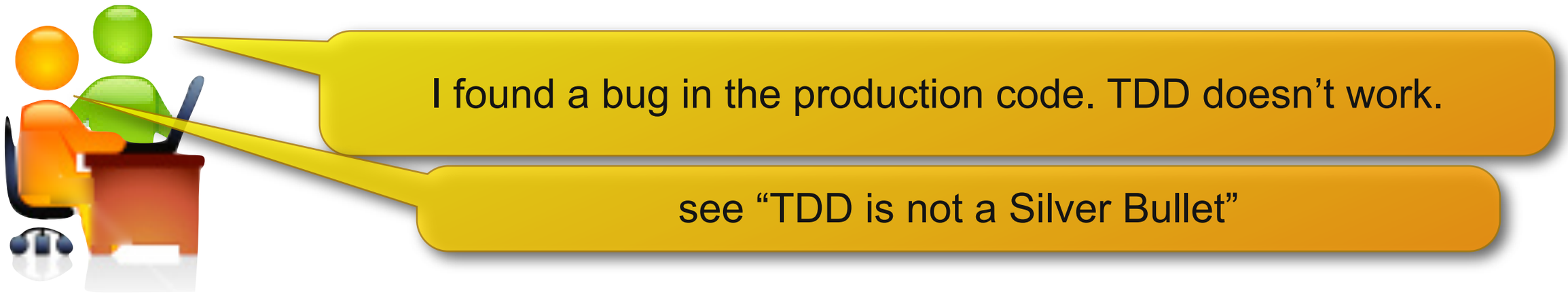
TDD for unbelievers

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

Treat your test code as production code

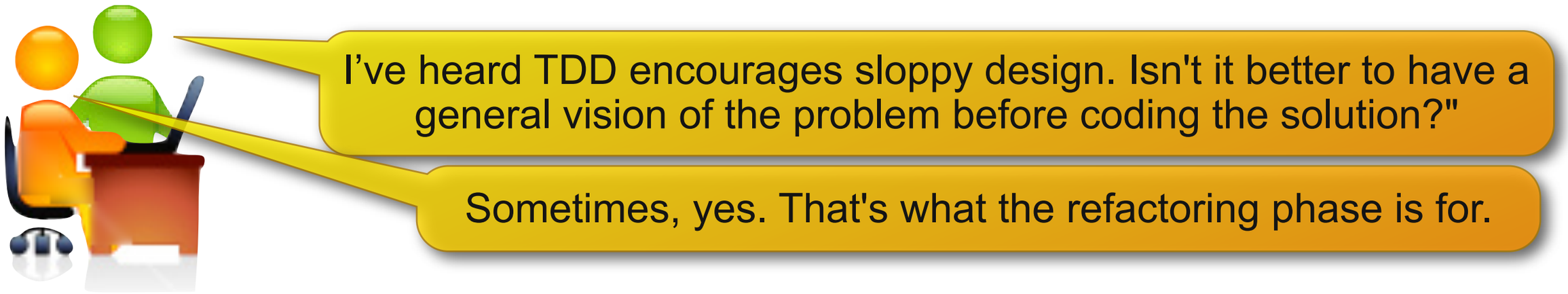
- Be wary of test maintenance overhead
- Test code is not second-class code
- Refactor to keep your test code maintanable
- Avoid technical dept

TDD for unbelievers



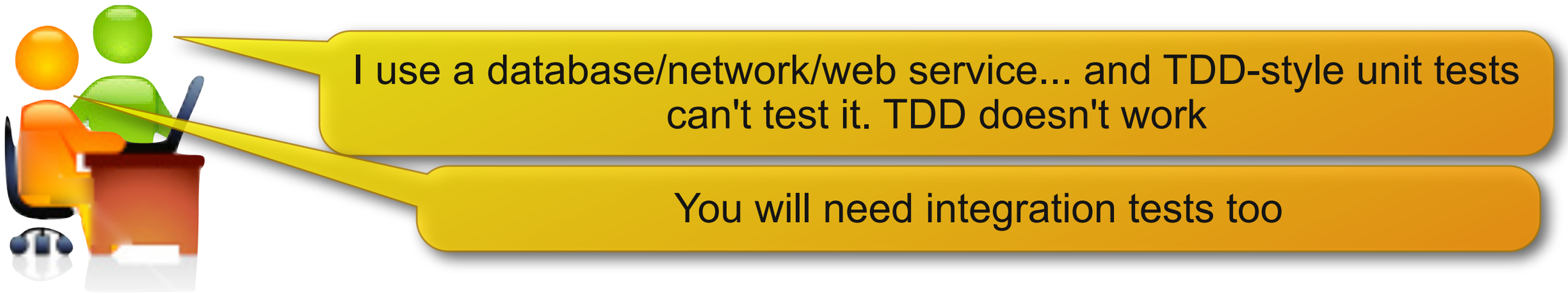
- You still need your testers!
- You can still make mistakes...
- ...but they are easier to isolate and to fix
- If you find a bug, just add another test to reproduce it, then fix it!

TDD for unbelievers



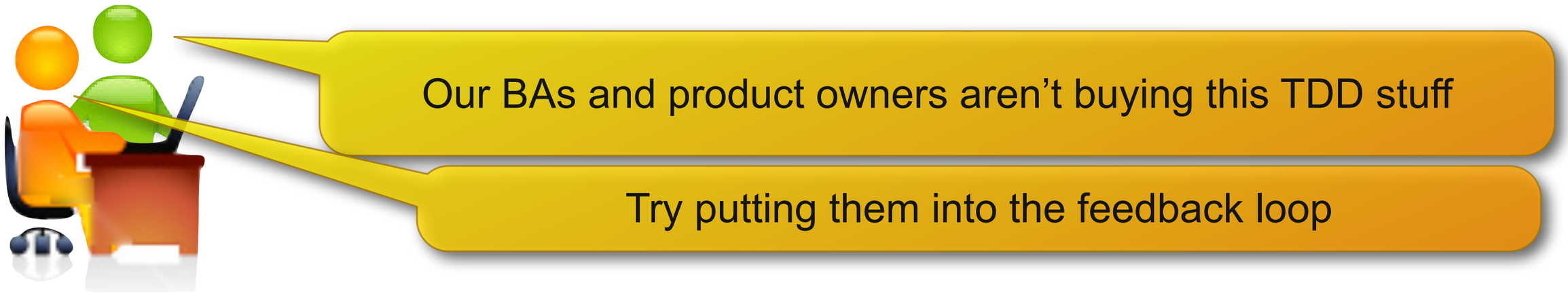
- TDD does not preclude initial high-level design
  - Brainstorm a high-level design approach
  - Define a common well-known domain language
  - Use an ‘Iteration 0’ and spikes
  - Don’t be afraid to refactor
- TDD works better when you have a vision

TDD for unbelievers



- 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

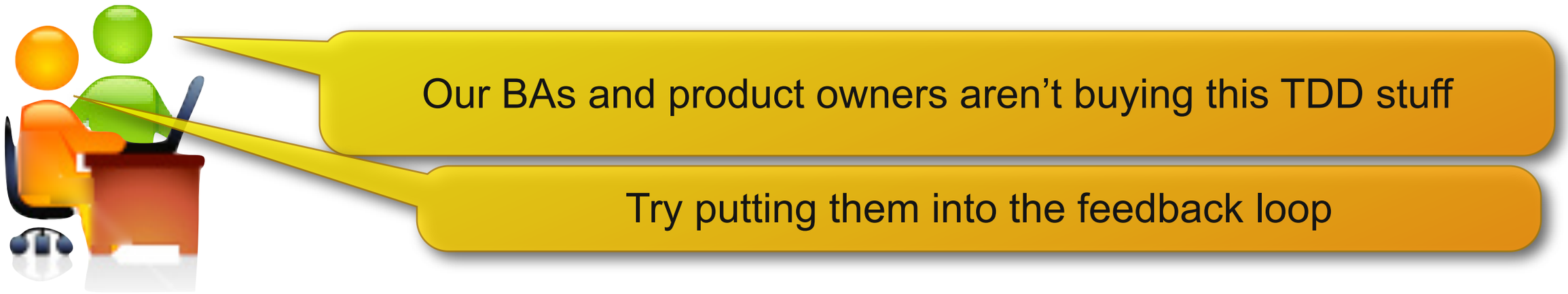
TDD for unbelievers



- 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

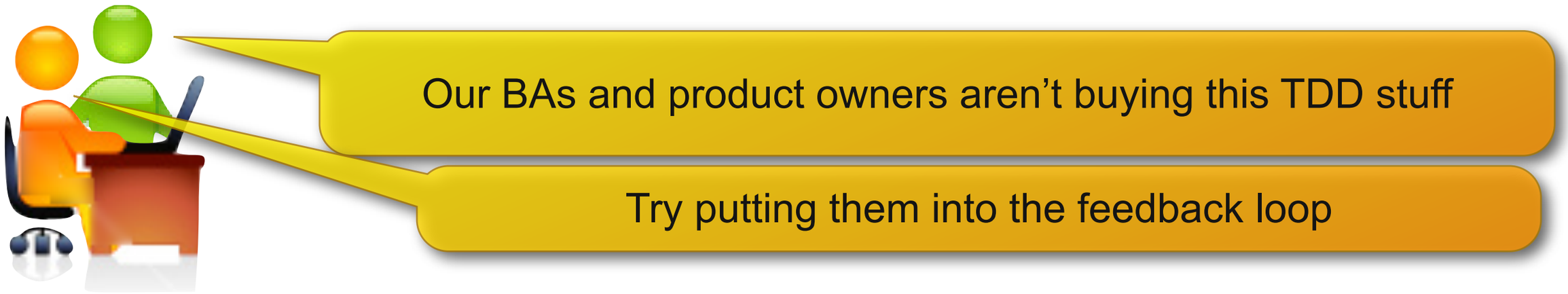


- Acceptance-Test Driven Development is about communication
- BDD help BAs and POs participate in writing executable requirements

```
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 ""...
                                  ...
                                  ..."" 
    }
}
```

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

TDD for unbelievers

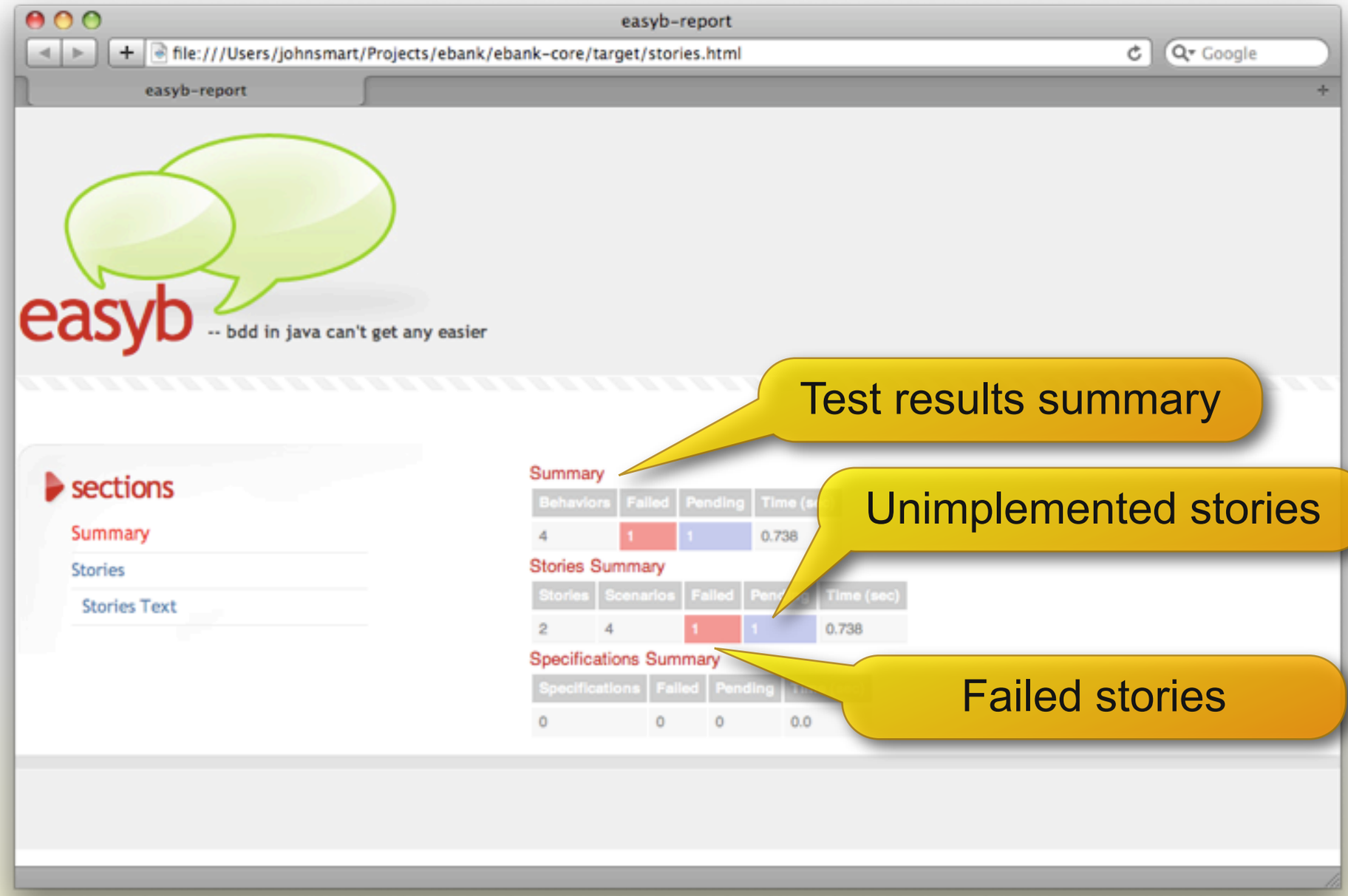


- Acceptance-Test Driven Development is about communication
- BDD help BAs and POs participate in writing executable requirements

```
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

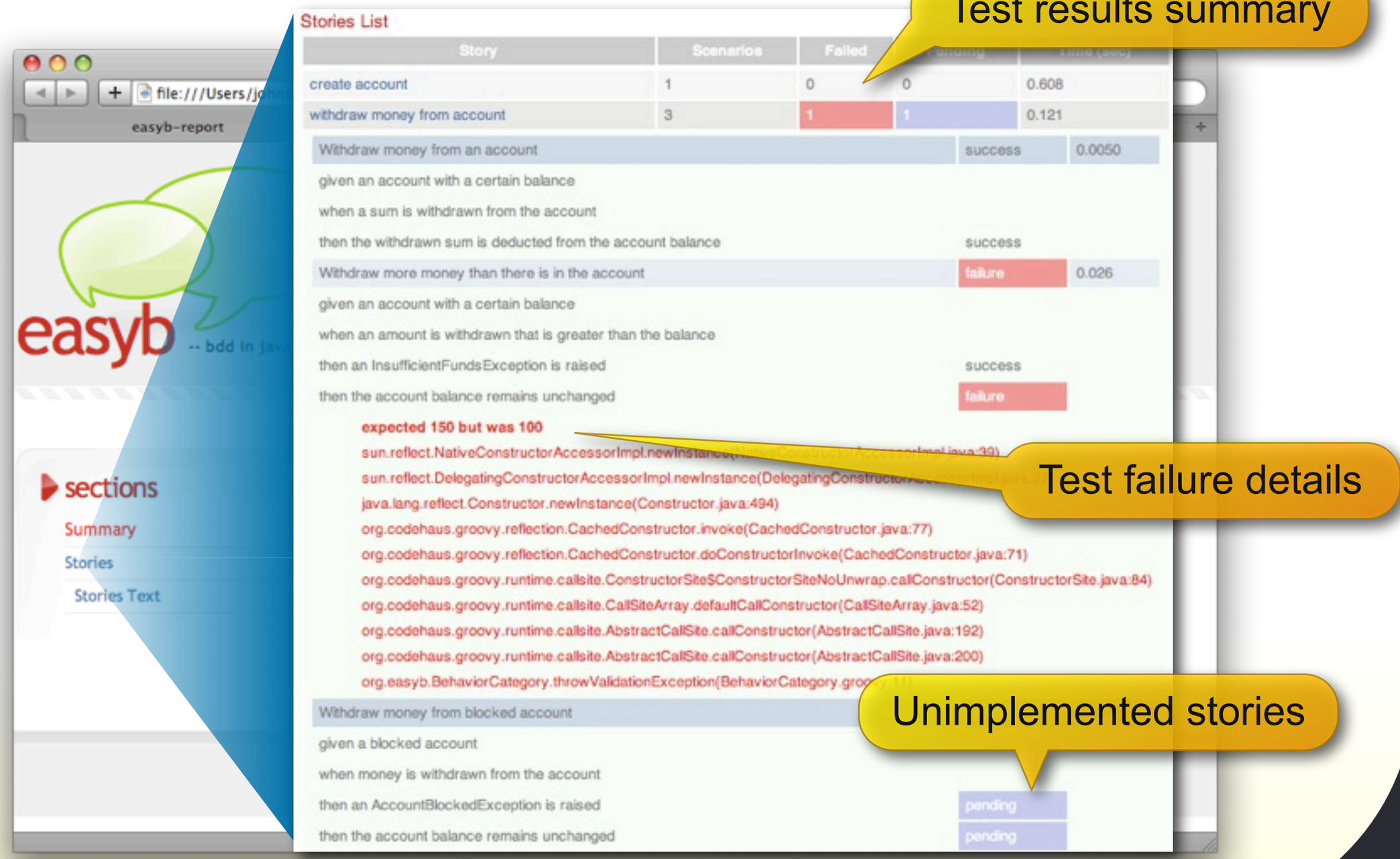
Many BDD tools also have great reporting features



TDD for unbelievers

Many BDD tools have great reporting features

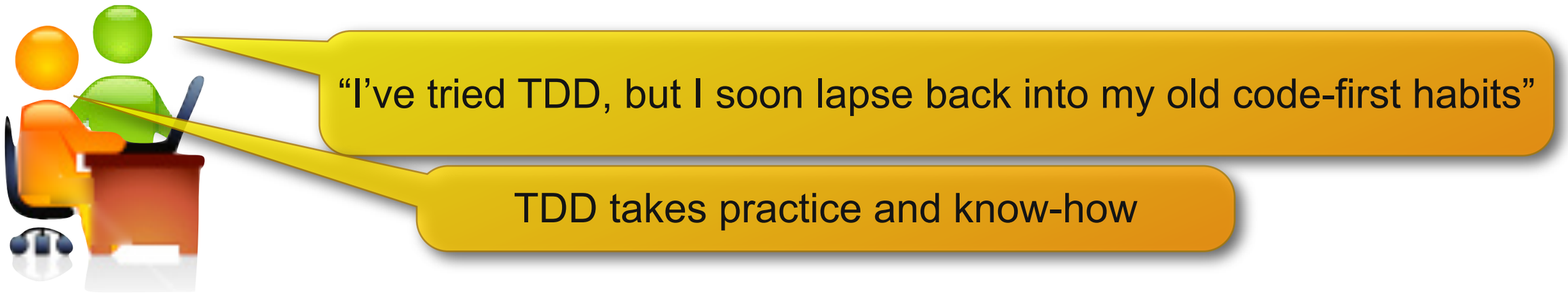
Test results summary



Test failure details

Unimplemented stories

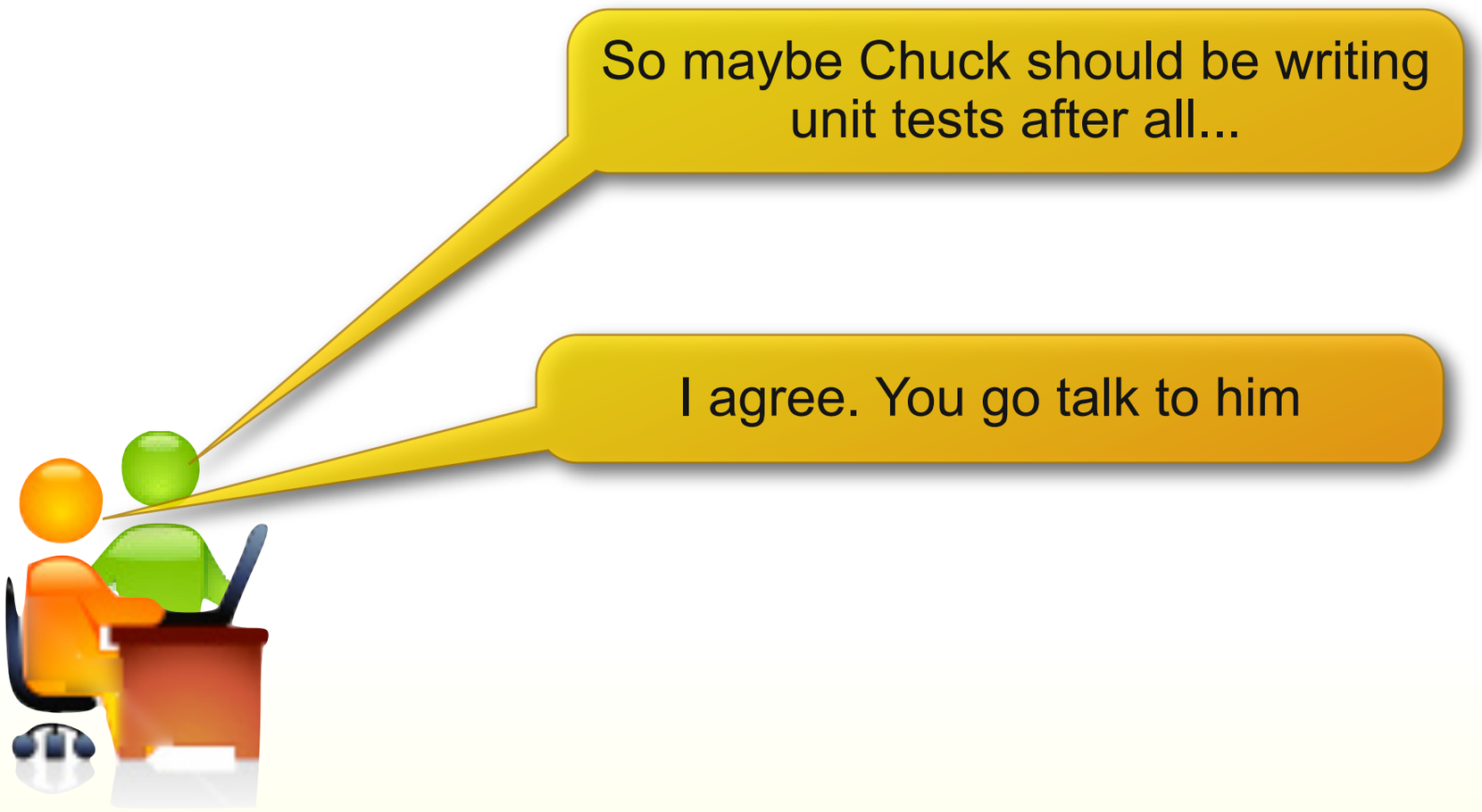
TDD for unbelievers



- 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?



Thank You

John Ferguson Smart
Wakaleo Consulting Ltd.

http://www.wakaleo.com

Email: john.smart@wakaleo.com
Twitter: wakaleo