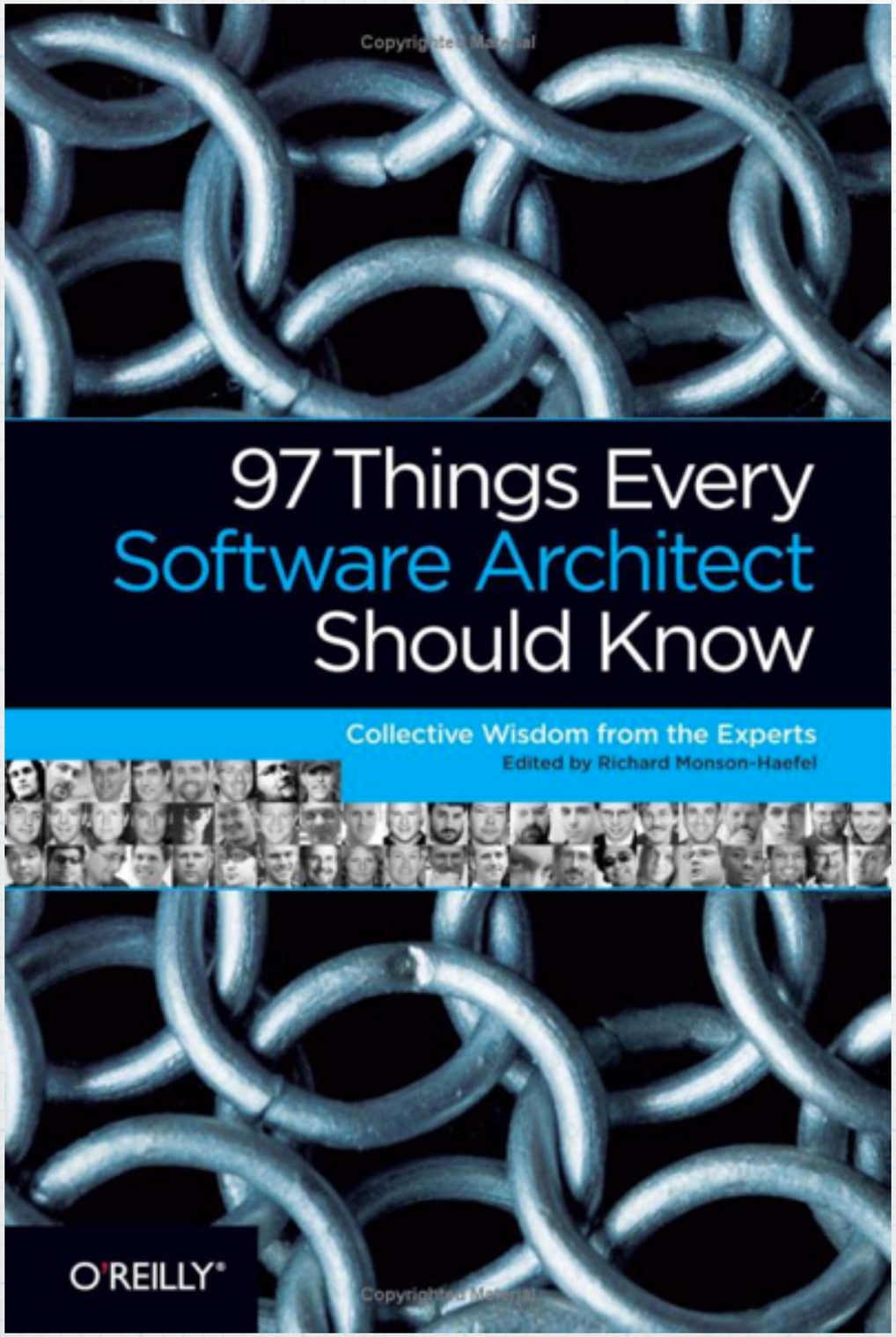
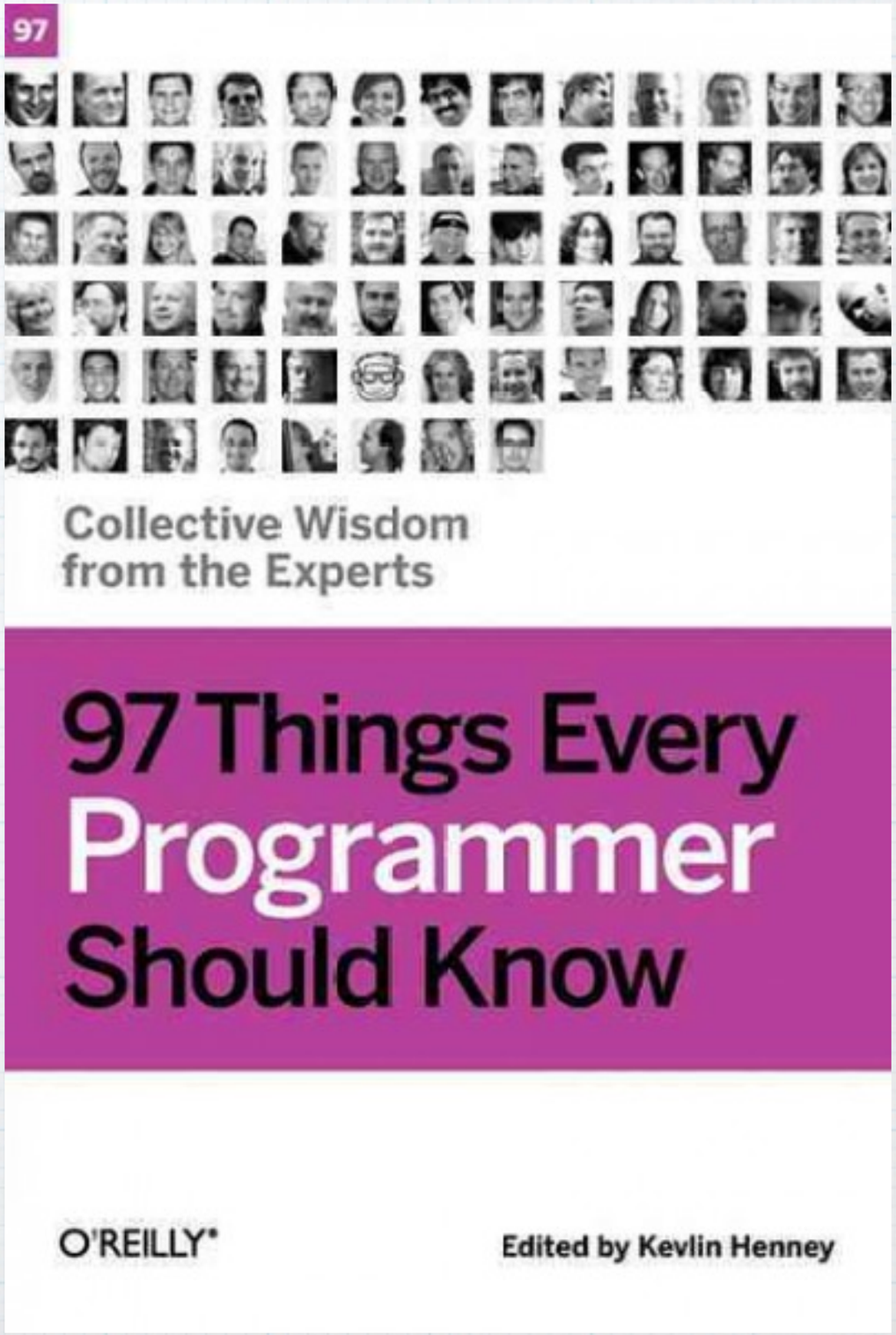
Building Uncle Bob's Button

Burk Hufnagel - Technical Architect
Daugherty Business Solutions

Who is Burk Hufnagel?





Who is Burk Hufnagel?

Technical reviewer for SCJP 6 Study Guide.
"Burk fixed more code than we care to admit."
— Kathy Sierra & Bert Bates

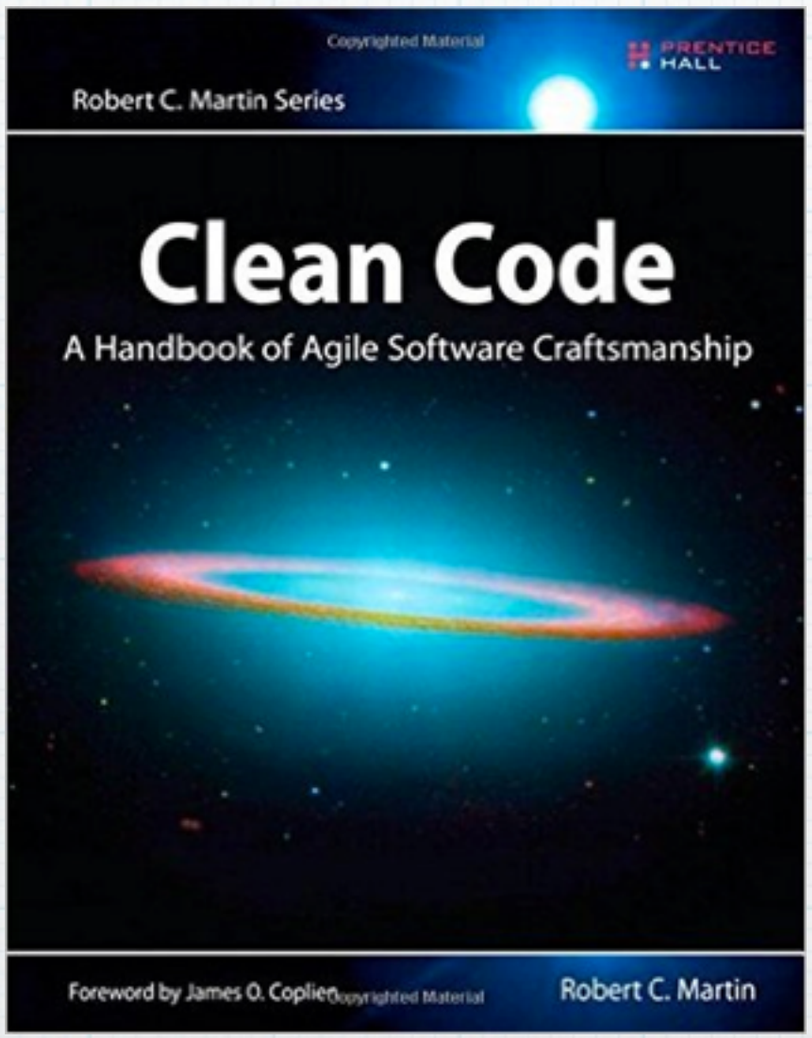
Sun Certified Java Programmer, Developer and Enterprise Architect.

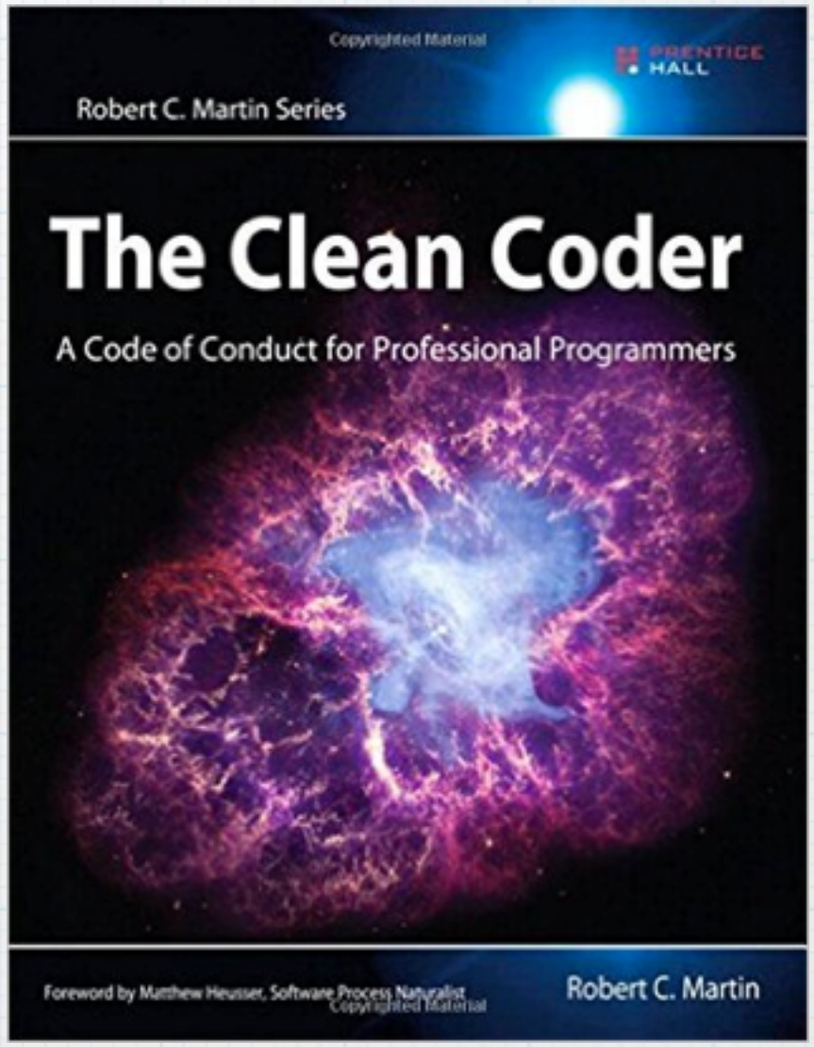
Programmer and Architect with Daugherty Business Solutions.

Who is "Uncle Bob"?

Robert C. Martin
aka Uncle Bob

Co-signer of the Agile Manifesto





Advocate of Test Driven Development.

What is Uncle Bob's Button?

Sometime before 2010,
a question was posted on Quora.com:

"Does test-driven development (TDD)
really improve software quality?"

On June 18, 2015, Robert C. Martin
(Uncle Bob), posted his answer...

What is Uncle Bob's Button?

Guns don't kill people.
People kill people.

TDD doesn't improve software quality.
People improve software quality.

But TDD, used responsibly,
with the right training, is an assault rifle.

What is Uncle Bob's Button?

Imagine having a push button that could tell you, in a few seconds, whether your whole system was working as expected or not.

Push it and it lights up green if everything is working, or red if it isn't...

What could you do, if you had that button?

How fast could you go?

(Paraphrased from the original by me)

Why Do You Want Uncle Bob's Button?

What could you do
if you had that button?

How fast could you go
if you had that button?

Buy vs Build

Why not find and buy a software package or subscribe to a service?

Turns out it's a custom job that has to happen while the software is being written...

Real World Version

For Uncle Bob's Button you would need:

* Button - press to initiate process
* Sensors - (custom) to examine and report
* Red and Green Lights - visual feedback
* Wire - connect everything together
* Power Source - used by sensors and lights

Software Version

In software, it's simpler but not easier:

* Test framework - wires and power source
* Unit tests - sensors and lights
* Follow Test Driven Development to create the unit tests

Wait: What's a Unit Test?

Unit test are automated tests focused on one piece of code; for Java this is often a single class.

It's not a unit test if it:

* Talks to the database
* Communicates across the network
* Touches the file system
* Can't run while any other unit tests are running
* Requires changing the environment
  (like editing config files) for it to run.

What is TDD?

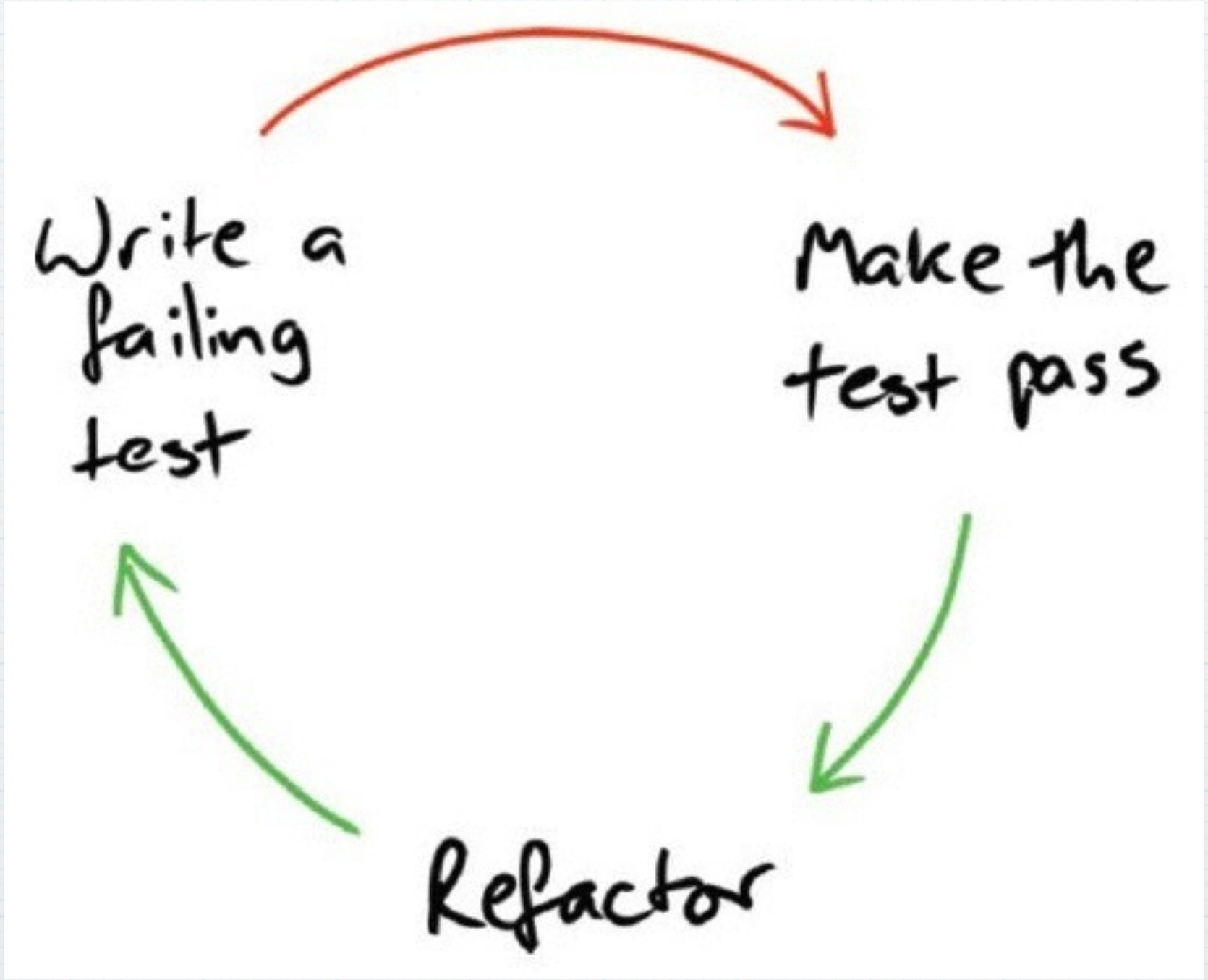


image from: a4direct.com

Building Uncle Bob's Button is a Do It Yourself project

* Unit Tests are written as you design and build your application.

* Unit Tests create value as you're writing them.

* There are problems, however...

A Common Concern

TDD: It's Not That Simple

By Gil Zilberfeld

> *"TDD is really easy to explain. The rules are easy to follow. Why then so few people actually do it, and when they do, so few actually do it well?*

We Only See Part of the Picture

www.motellevacancier.com

What Test Should You Write?

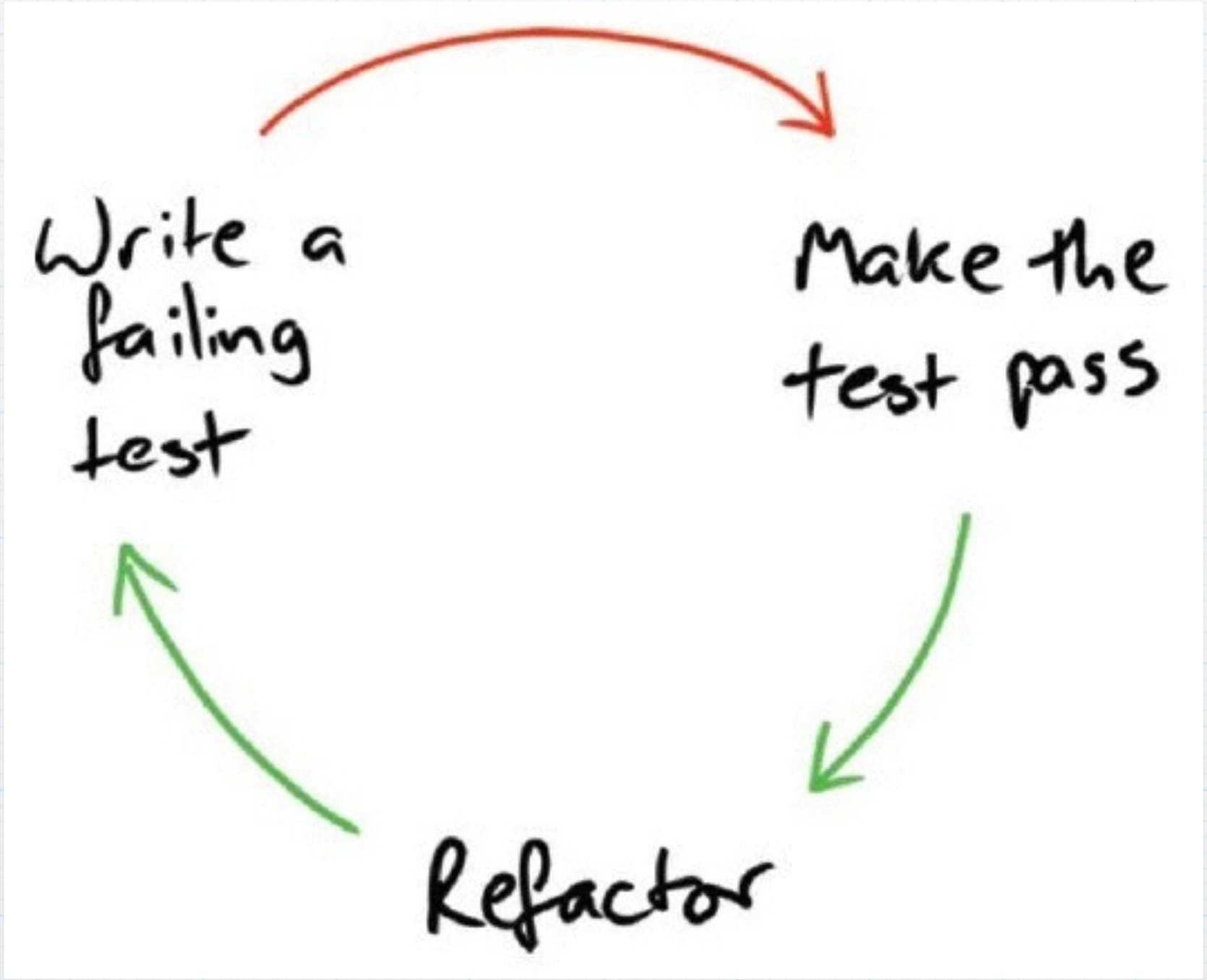


image from: a4direct.com

Uncle Bob's Button 2.0

Uncle Bob's Button is based on using Test Driven Development. This is Good News because TDD helps you design and build good code.

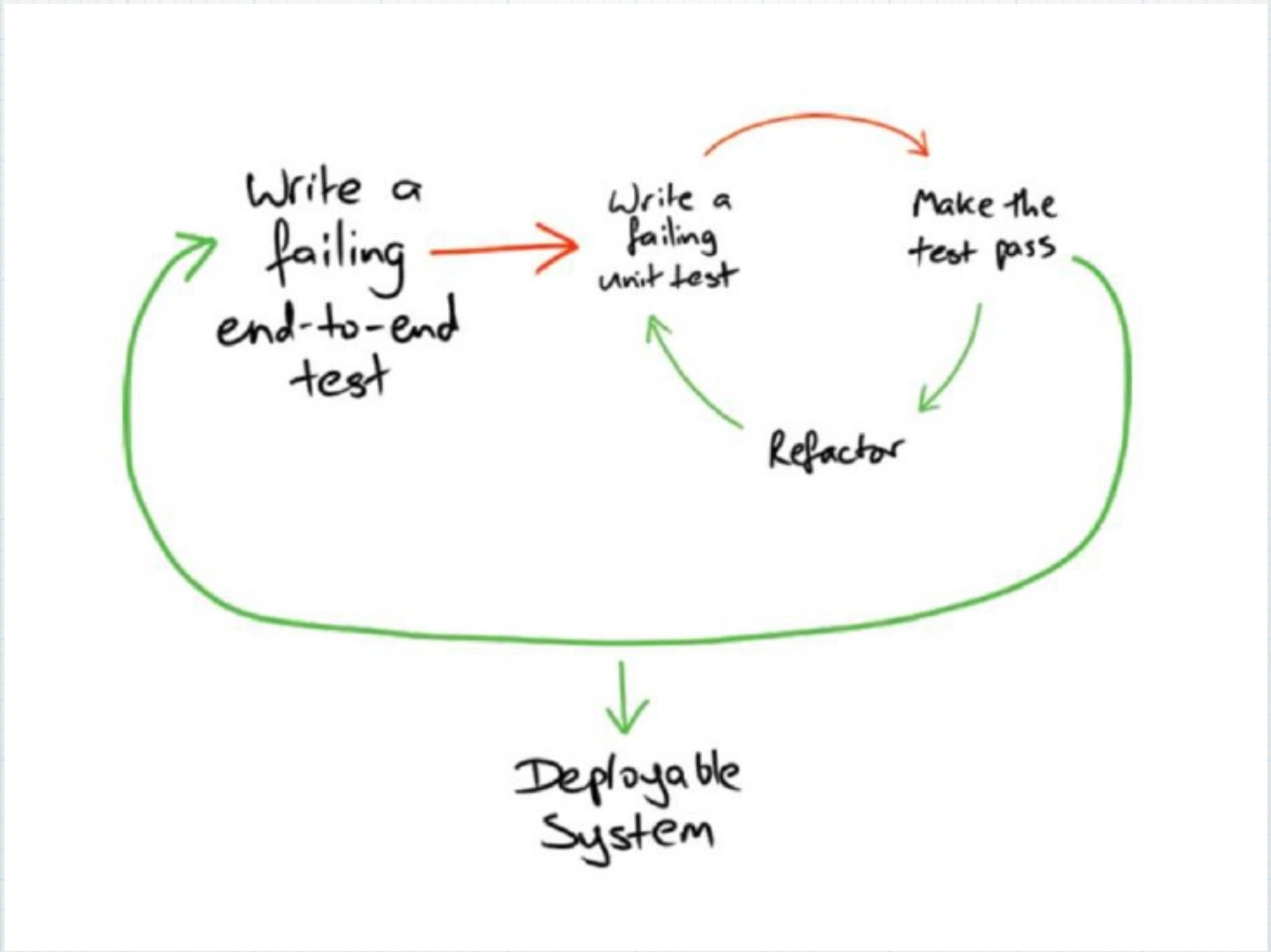
At the same time, TDD is not enough precisely because it is focused on units and an application is more that that.

Behavior Driven Development

Dan North
Father of BDD

"BDD is about implementing an application by describing its behavior from the perspective of its stakeholder."

Behavior Driven Development



images from: a4direct.com

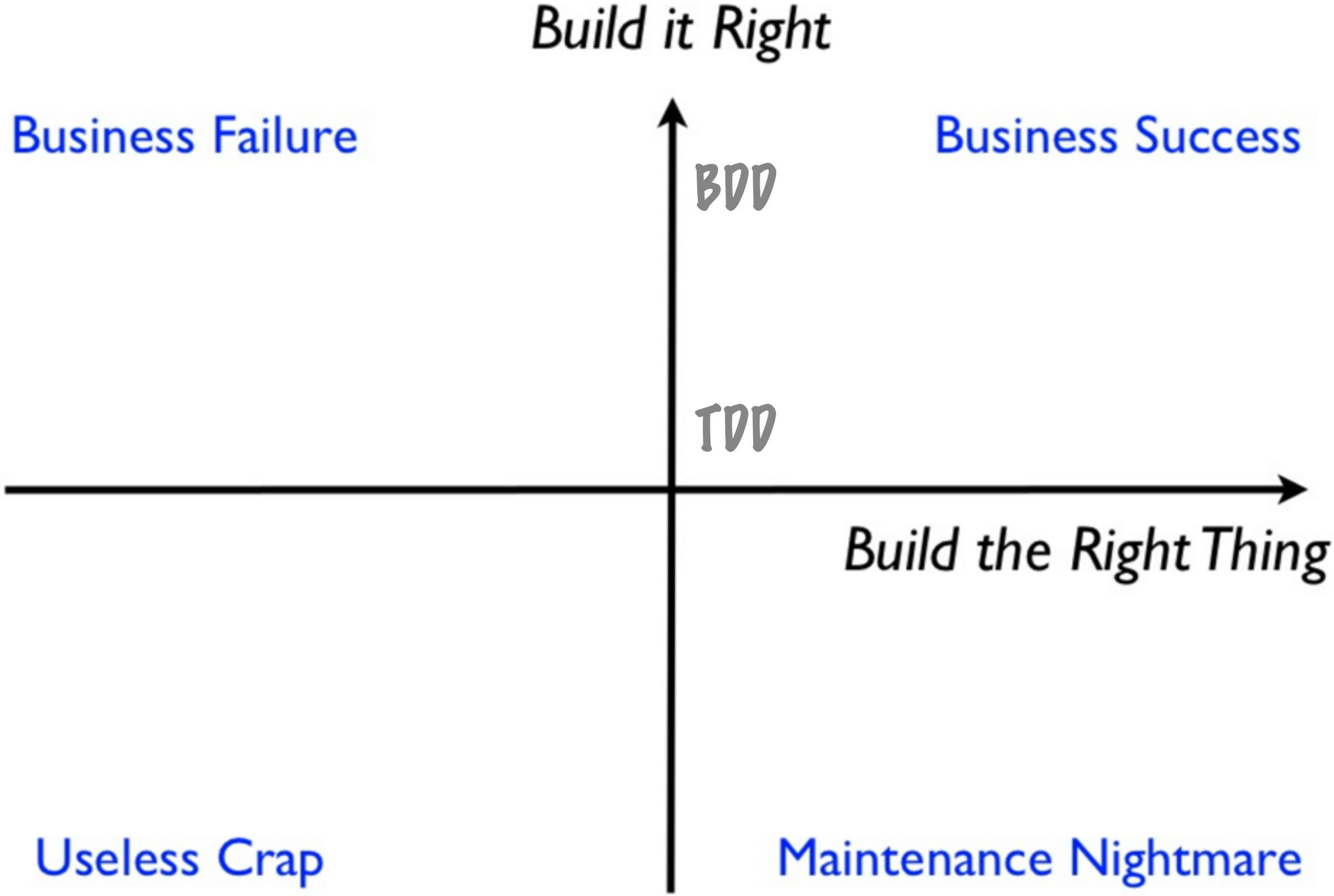
TDD vs BDD

TDD is about building things the right way

* High quality code
* Easy to understand, enhance and maintain

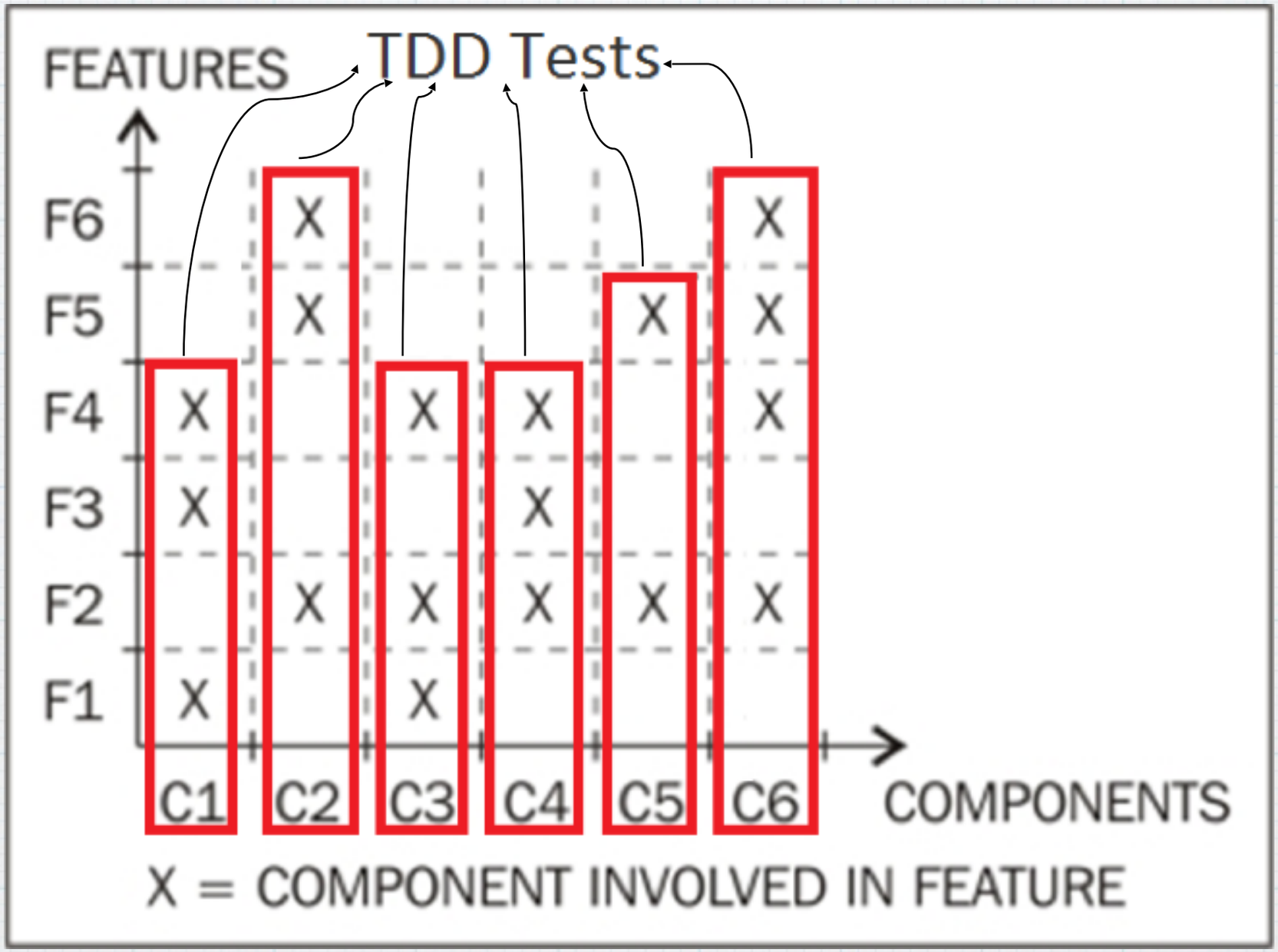
BDD is about building the right thing.

* Meeting business needs/requirements.



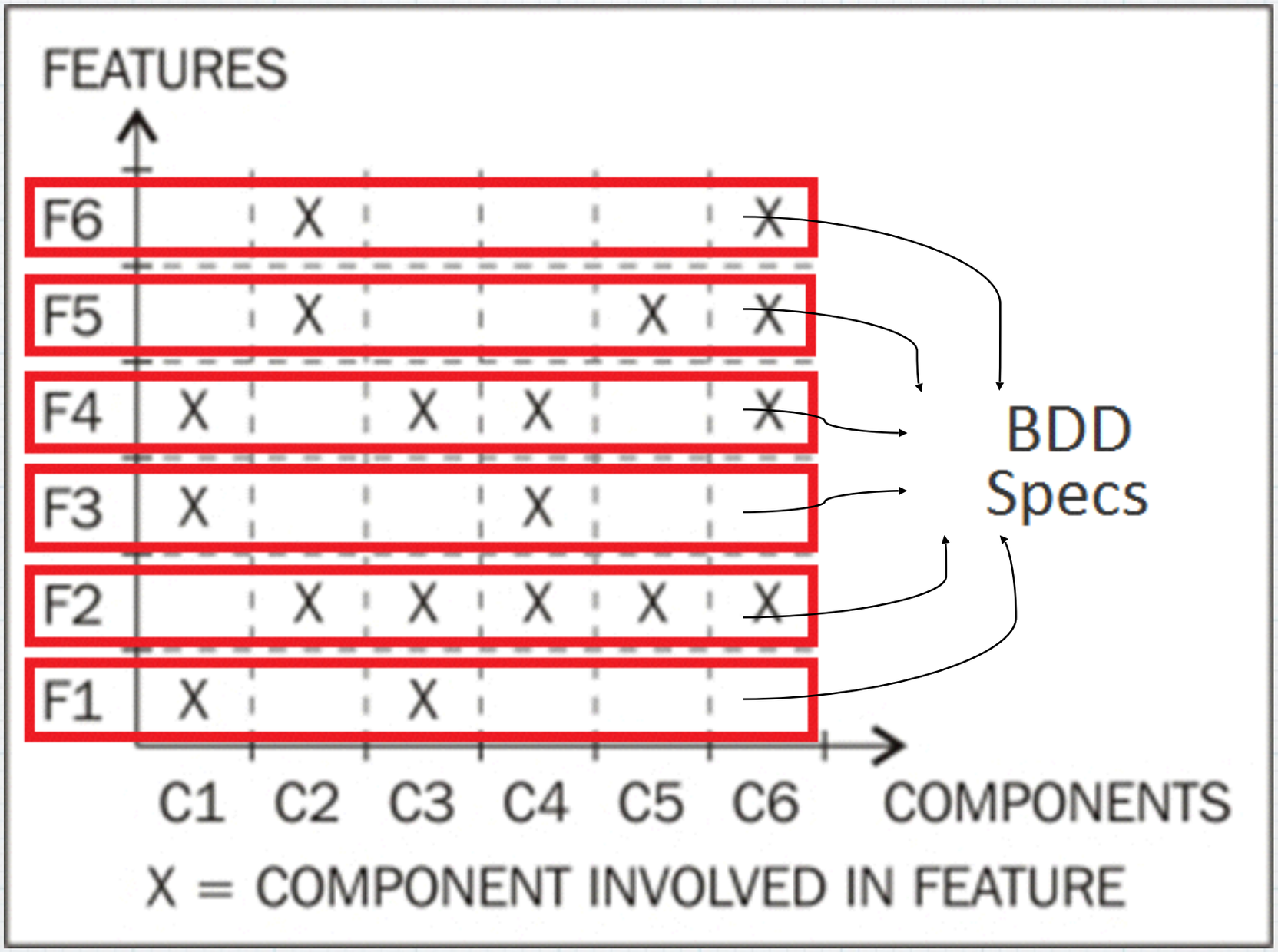
Specification By Example
Gojko Adzic, 2011 page 4

TDD's Focus



Modified diagram from "Learning Behaviour-driven Development with JavaScript"

BDD's Focus

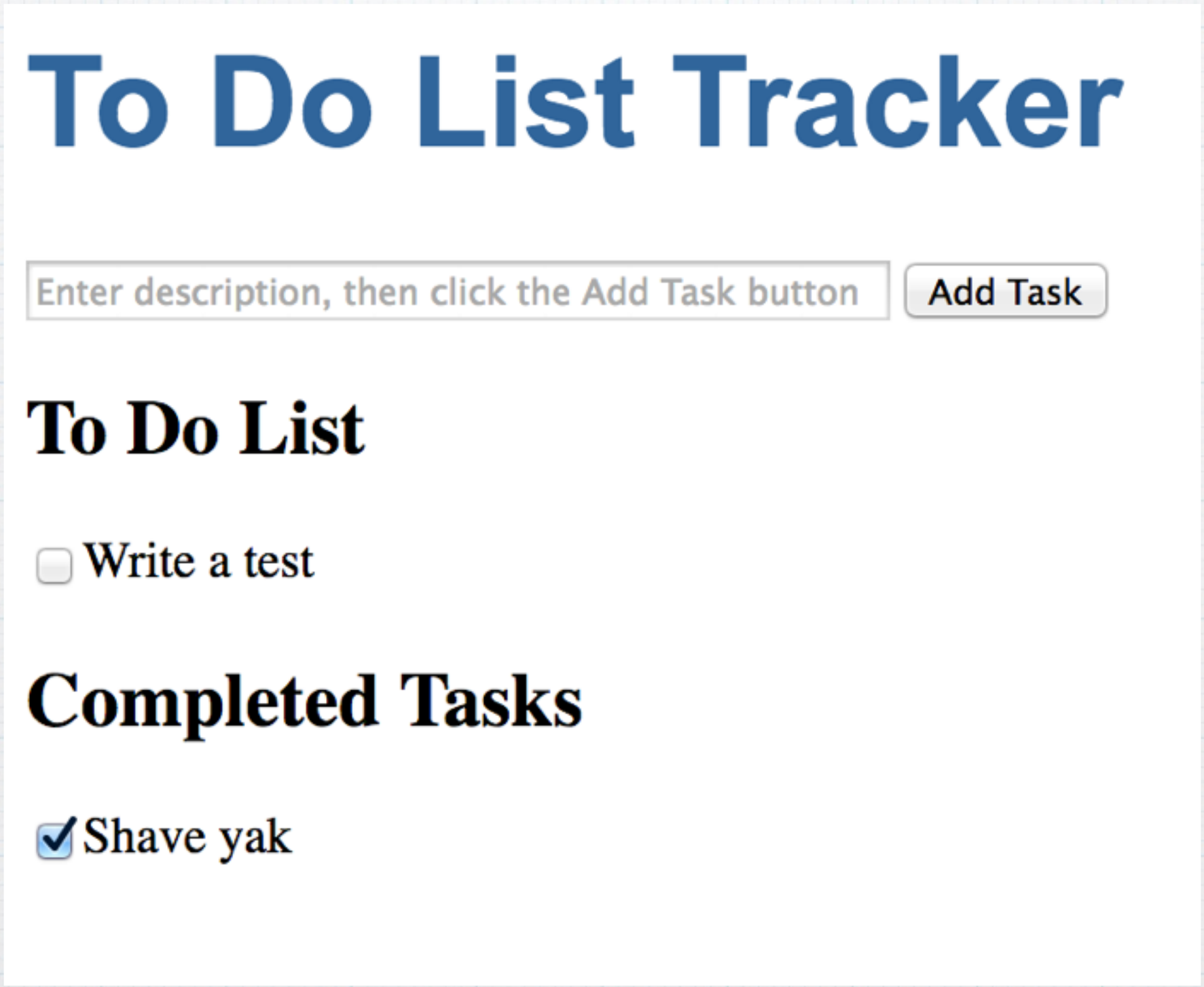


Modified diagram from "Learning Behaviour-driven Development with JavaScript"

My beliefs

Based on my experience,
I believe that Test Automation,
when used properly,
results in delivering better quality code
in less time.

Let's see it in action...



An Outside-In Approach

* Specification By Example/ Behavior Driven Development (Cucumber JS)
* Test Driven Development (JUnit)

To Do List Goal

As a human with too much on my mind

I want an online To Do List tracker

So that I don't have to remember it all

Create a Feature File

Feature: Add items to my To Do List
  As a human with too much on my mind
  I want to add items to my To Do List
  So that I don't have to remember them all

Create a Feature File

Scenario: Adding items

Given an empty list
When I add "Write a test" to the list
Then it should be displayed in the To Do area
When I add "Shave yak" to the list
Then it should be displayed in the To Do area
And it should be under "Write a test"

Cucumber-JS Results

```
bth0624$ ./node_modules/.bin/cucumber.js
Feature: Add items to my To Do List

  As a human with too much on my mind
  I want to add items to my To Do List
  So that I don't have to remember them all

  Scenario: Adding items
  ? Given an empty list
  ? When I add "Write a test" to the list
  ? Then it should be displayed in the To Do area
  ? When I add "Shave yak" to the list
  ? Then it should be displayed in the To Do area
  ? And it should be under "Write a test"

Warnings:

1) Scenario: Adding items - features/ToDoList.feature:7
   Step: Given an empty list - features/ToDoList.feature:8
   Message:
     Undefined. Implement with the following snippet:

       this.Given(/^an empty list$/, function (callback) {
         // Write code here that turns the phrase above into concrete actions
         callback(null, 'pending');
       });

2) Scenario: Adding items - features/ToDoList.feature:7
```

Create Steps Defn file

```
module.exports = function() {
    this.Given(/^an empty list$/, function () {
        return 'pending';
    });

    this.When(/^I add “Write a test” to the list$/, function () {
        return 'pending';
    });

    this.Then(/^it should be displayed in the To Do area$/, function () {
        return 'pending';
    });

    this.When(/^I add “Shave yak” to the list$/, function () {
        return 'pending';
    });

    this.Then(/^it should be under “Write a test”$/, function () {
        return 'pending';
    });
};
```

Create Steps Defn file

```
module.exports = function() {
    this.Given(/^an empty list$/, function () {
        return 'pending';
    });

    this.When(/^I add “Write a test” to the list$/, function () {
        return 'pending';
    });

    this.Then(/^it should be displayed in the To Do area$/, function () {
        return 'pending';
    });

    this.When(/^I add “Shave yak” to the list$/, function () {
        return 'pending';
    });

    this.Then(/^it should be under “Write a test”$/, function () {
        return 'pending';
    });
};
```

Create Steps Defn file

```
module.exports = function() {
    this.Given(/^an empty list$/, function () {
        return 'pending';
    });

    this.When(/^I add “(.*)” to the list$/, function (task) {
        return 'pending';
    });

    this.Then(/^it should be displayed in the To Do area$/, function () {
        return 'pending';
    });

    this.Then(/^it should be under “(.*)”$/, function (task) {
        return 'pending';
    });
};
```

Cucumber-JS Results

```
bth0624$ ./node_modules/.bin/cucumber.js
Feature: Add items to my To Do List

  As a human with too much on my mind
  I want to add items to my To Do List
  So that I don't have to remember them all

  Scenario: Adding items
  ? Given an empty list
  - When I add "Write a test" to the list
  - Then it should be displayed in the To Do area
  - When I add "Shave yak" to the list
  - Then it should be displayed in the To Do area
  - And it should be under "Write a test"
Warnings:

1) Scenario: Adding items - features/ToDoList.feature:7
   Step: Given an empty list - features/ToDoList.feature:8
   Step Definition: features/step-definitions/ToDoList.step_def.js:9
   Message:
     Pending

1 scenario (1 pending)
7 steps (1 pending, 6 skipped)
0m00.001s
```

Update Steps Defn file

```
module.exports = function() {
    this.When(/^I add “(.*)” to the list$/, function (taskDescription) {
        this.taskAdded = taskDescription;
        return mainPage.addTask(taskDescription);
    });

    this.Then(/^it should be displayed in the To Do area$/, function () {
        return expect(mainPage.toDoListContains(this.taskAdded)).to.be.true;
    });

    this.Then(/^it should be under “(.*)”$/, function (task) {
        let indexOne = mainPage.indexOfTask(task);
        let indexTwo = mainPage.indexOfTask(this.taskAdded);
        return expect(indexOne).to.be.(indexTwo-1);
    });
};
```

Server Side

* Web Service used by the To Do Client
    * Methods to add a task, find a task by index, and get the task list
* Use TDD to design the classes

Uncle Bob's Three Laws

1. You may not write any production code until you have first written a failing unit test.

2. You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

3. You may not write more production code than is sufficient to pass the currently failing unit test.

Uncle Bob's Three Laws

in action...

A scenario describes high level behavior.

```
Scenario: Adding items
  Given an empty list
   When I add "Write a test" to the list
    Then it should be displayed in the To Do area
    When I add "Shave yak" to the list
    Then it should be displayed in the To Do area
    And it should be under "Write a test"
```

Step definitions describe lower level behavior...

```
this.When(/^I add “Shave yak” to the list$/, function () {
    todoList.addTask( “Shave yak” )
});

this.Then(/^it should be displayed in the To Do area$/, function () {
    todoList.shouldDisplayTask( “Shave yak” );
});
```

which might suggests a unit test.

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
    }
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
    }
```

```
public class ListTracker {

}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
    }
```

```
public class ListTracker {

}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave yak" );
    }
```

```
public class ListTracker {

}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave yak" );
    }
```

```
public class ListTracker {
    public void addTask( String taskDescription ) {

    }
}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave yak" );
    }
```

```
public class ListTracker {
    public void addTask( String taskDescription ) {
        <= Does nothing. This should cause a failure later…
    }

}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();
    }
```

```
public class ListTracker {
    public void addTask( String taskDescription ) {

    }
}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();
    }
```

```
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {

    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();
    }
```

```java
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {

    }
}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();
    }
```

```
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {
        return null; <== Explicitly invalid result
    }
}
```

```java
public class ListTrackerSpec {
   @Test
   public void taskAddedToListShouldBeRetrievable() {
      ListTracker listTracker = new ListTracker();
      listTracker.addTask( "Shave Yak" );

      List<String> tasks = listTracker.getTasks();
   }
```

```java
public class ListTracker {
   public void addTask( String taskDescription ) {

   }

   public List<String> getTasks() {
      return null;
   }
}
```

```java
public class ListTrackerSpec {
   @Test
   public void taskAddedToListShouldBeRetrievable() {
      ListTracker listTracker = new ListTracker();
      listTracker.addTask( "Shave Yak" );

      List<String> tasks = listTracker.getTasks();

      assertTrue( tasks.contains("Shave Yak") );
   }
```

```java
public class ListTracker {
   public void addTask( String taskDescription ) {

   }

   public List<String> getTasks() {
      return null;
   }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

Console | Markers | Progress | JUnit

Finished after 0.034 seconds

Runs: 1/1 | Errors: 1 | Failures: 0

com.burkhufnagel.ListTrackerSpec [Runner: JUnit 4] (0.001 s)

taskAddedToListShouldBeRetrievable (0.001 s)

Failure Trace

java.lang.NullPointerException

at com.burkhufnagel.ListTrackerSpec.taskAddedToListShouldBeRetrievab

```java
    }

    public List<String> getTasks() {
        return null;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {
        return null;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {
        String[] todos = {”Shave Yak”};
        return Arrays.asList( todos );
    }
}
```

```
public class ListTrackerSpec {
   @Test
   public void taskAddedToListShouldBeRetrievable() {
      ListTracker listTracker = new ListTracker();
      listTracker.addTask( "Shave Yak" );

      List<String> tasks = listTracker.getTasks();

      assertTrue( tasks.contains("Shave Yak") );
   }
```

```
public class ListTracker {
   public void addTask( String taskDescription ) {

   }

   public List<String> getTasks() {
      return new ArrayList<String>();
   }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

Console | Markers | Progress | JUnit

Finished after 0.019 seconds

Runs: 1/1 | Errors: 0 | Failures: 1

com.burkhufnagel.ListTrackerSpec [Runner: JUnit 4] (0.000 s)

taskAddedToListShouldBeRetrievable (0.000 s)

Failure Trace

java.lang.AssertionError

at com.burkhufnagel.ListTrackerSpec.taskAddedToListShouldBeRetrievab

```java
    }

    public List<String> getTasks() {
        return new ArrayList<String>();
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {
        return new ArrayList<String>();
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    private List<String> tasks = new ArrayList<String>();

    public void addTask( String taskDescription ) {

    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    private List<String> tasks = new ArrayList<String>();

    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

Console | Markers | Progress | JUnit

Finished after 0.022 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

com.burkhufnagel.ListTrackerSpec [Runner: JUnit 4] (0.001 s)

Failure Trace

```java
    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    private List<String> tasks = new ArrayList<String>();

    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```java
public class ListTracker {
    private List<String> tasks = new ArrayList<String>();

    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        String taskDescription = "Shave Yak";
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( "Shave Yak" );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains("Shave Yak") );
    }
```

```
public class ListTracker {
    private List<String> tasks = new ArrayList<String>();

    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

```
public class ListTrackerSpec {
   @Test
   public void taskAddedToListShouldBeRetrievable() {
      String taskDescription = "Shave Yak";
      ListTracker listTracker = new ListTracker();
      listTracker.addTask( taskDescription );

      List<String> tasks = listTracker.getTasks();

      assertTrue( tasks.contains( taskDescription ) );
   }
```

```
public class ListTracker {
   private List<String> tasks = new ArrayList<String>();

   public void addTask( String taskDescription ) {
      this.tasks.add( taskDescription );
   }

   public List<String> getTasks() {
      return tasks;
   }
}
```

```java
public class ListTrackerSpec {
    @Test
    public void taskAddedToListShouldBeRetrievable() {
        String taskDescription = "Shave Yak";
        ListTracker listTracker = new ListTracker();
        listTracker.addTask( listTracker );

        List<String> tasks = listTracker.getTasks();

        assertTrue( tasks.contains(listTracker) );
```

Console | Markers | Progress | JUnit

Finished after 0.022 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

com.burkhufnagel.ListTrackerSpec [Runner: JUnit 4] (0.001 s)

Failure Trace

```java
    public void addTask( String taskDescription ) {
        this.tasks.add( taskDescription );
    }

    public List<String> getTasks() {
        return tasks;
    }
}
```

Cucumber-JS Results

```
ToDoListClient$ ./node_modules/.bin/cucumber.js
Feature: Add items to my To Do List

  As a human with too much on my mind
  I want to add items to my To Do List
  So that I don’t have to remember them all

  Scenario: Adding items
  ✔ Given an empty list
  ✔ When I add “Write a test” to the list
  ✔ Then it should be displayed in the To Do area
  ✔ When I add “Shave yak” to the list
  ✔ Then it should be displayed in the To Do area
  ✔ And it should be under “Write a test”
```

Resources

"Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin

"Growing Object-Oriented Software, Guided by Tests" by Steve Freeman and Nat Pryce

"Specification By Example" by Gojko Adzic

"BDD in Action" by John Ferguson Smart

Questions & Contact Info

Feedback, questions, comments to:
burk.hufnagel@daugherty.com

Twitter: @burkhufnagel

Come to the monthly Atlanta Java User Group meetings. Info at ajug.org)