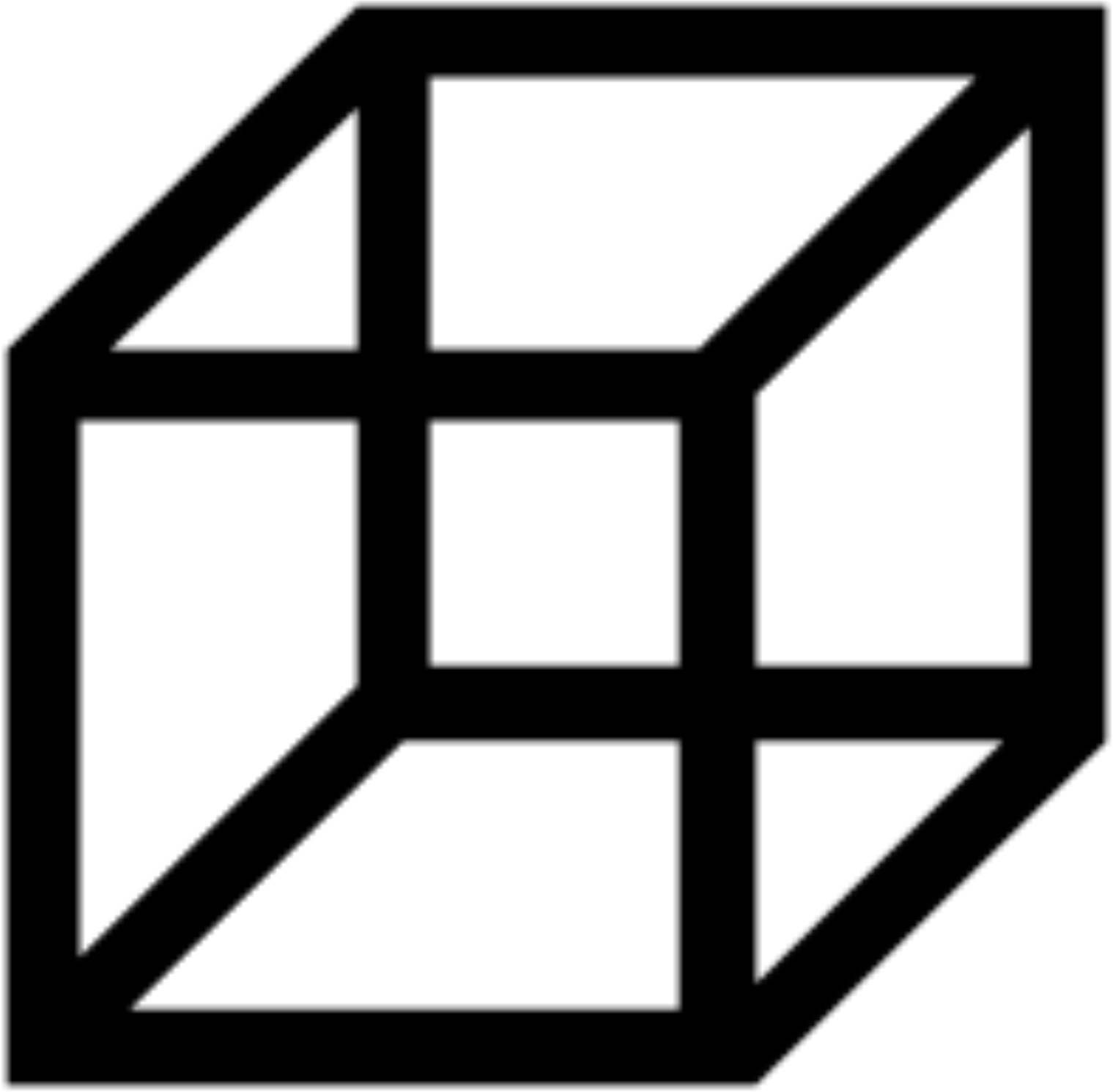
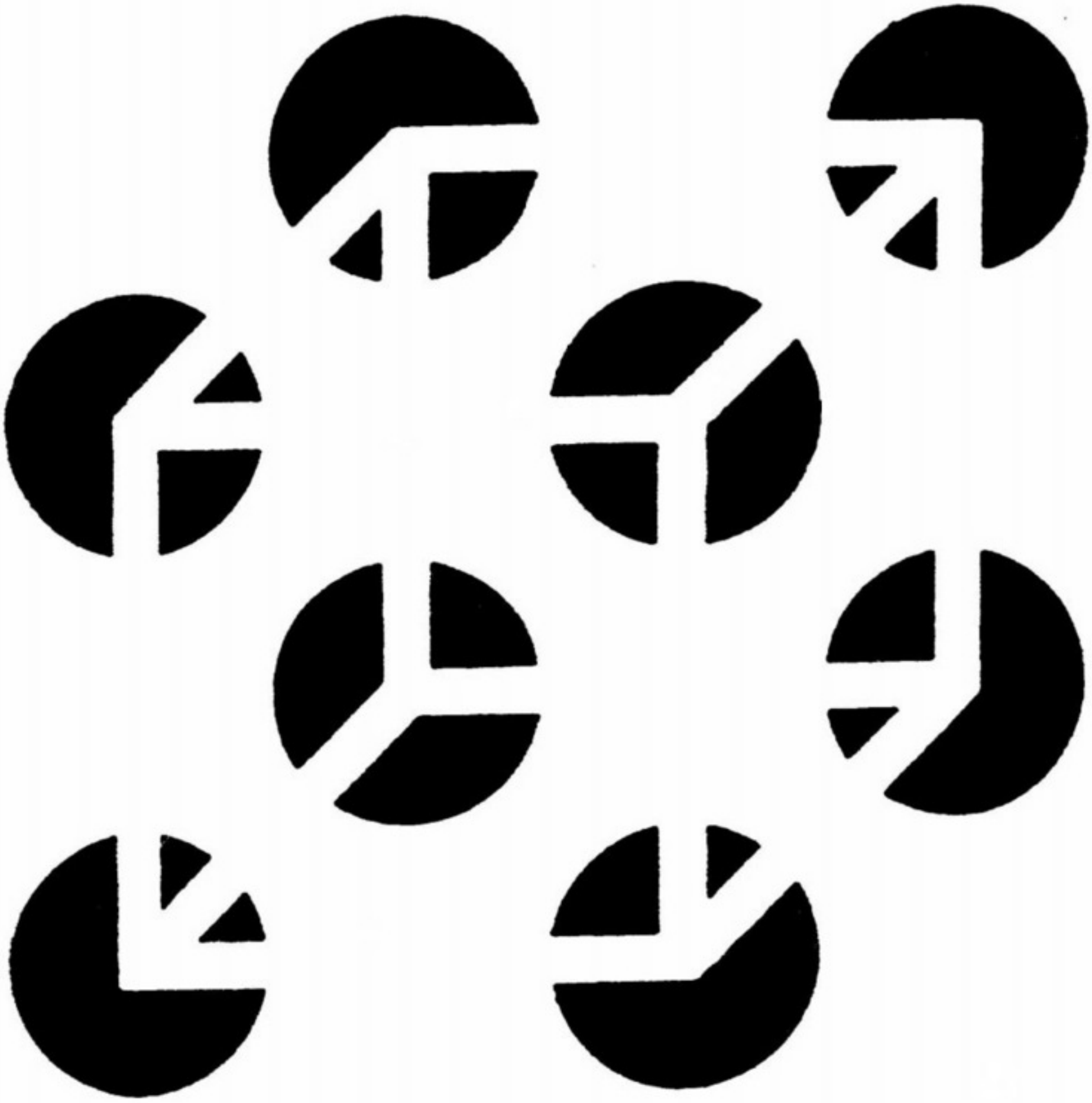
Delivering Better Code, Faster, with Spock [CON6182]

Burk Hufnagel - Technical Architect
Daugherty Business Solutions



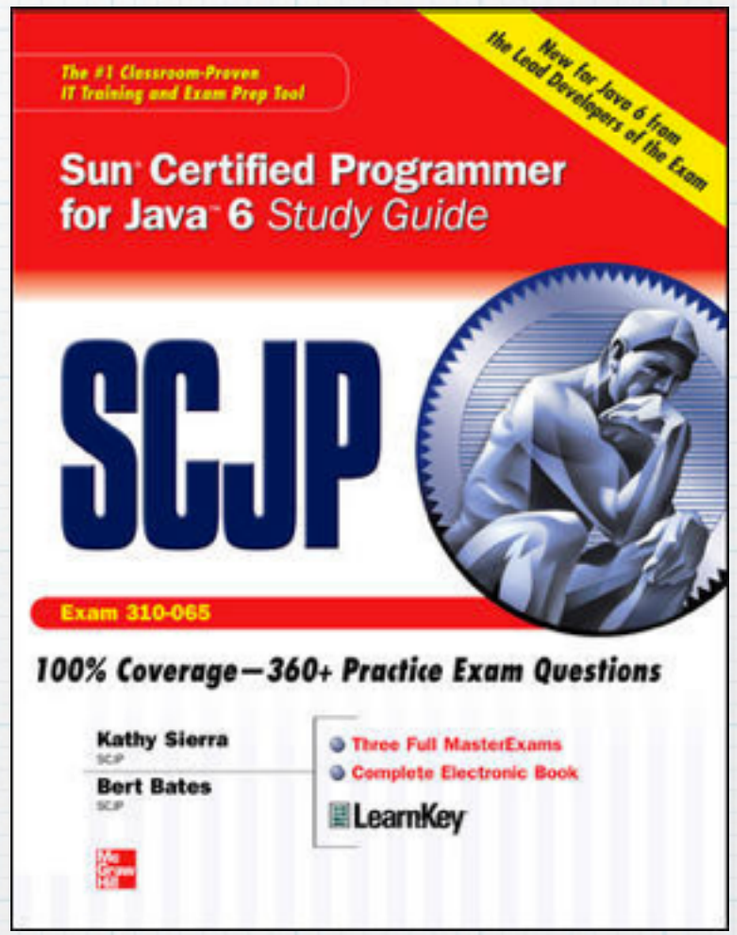


Delivering Better Code, Faster, with Spock [CON6182]

Burk Hufnagel - Technical Architect
Daugherty Business Solutions

Who is Burk Hufnagel?

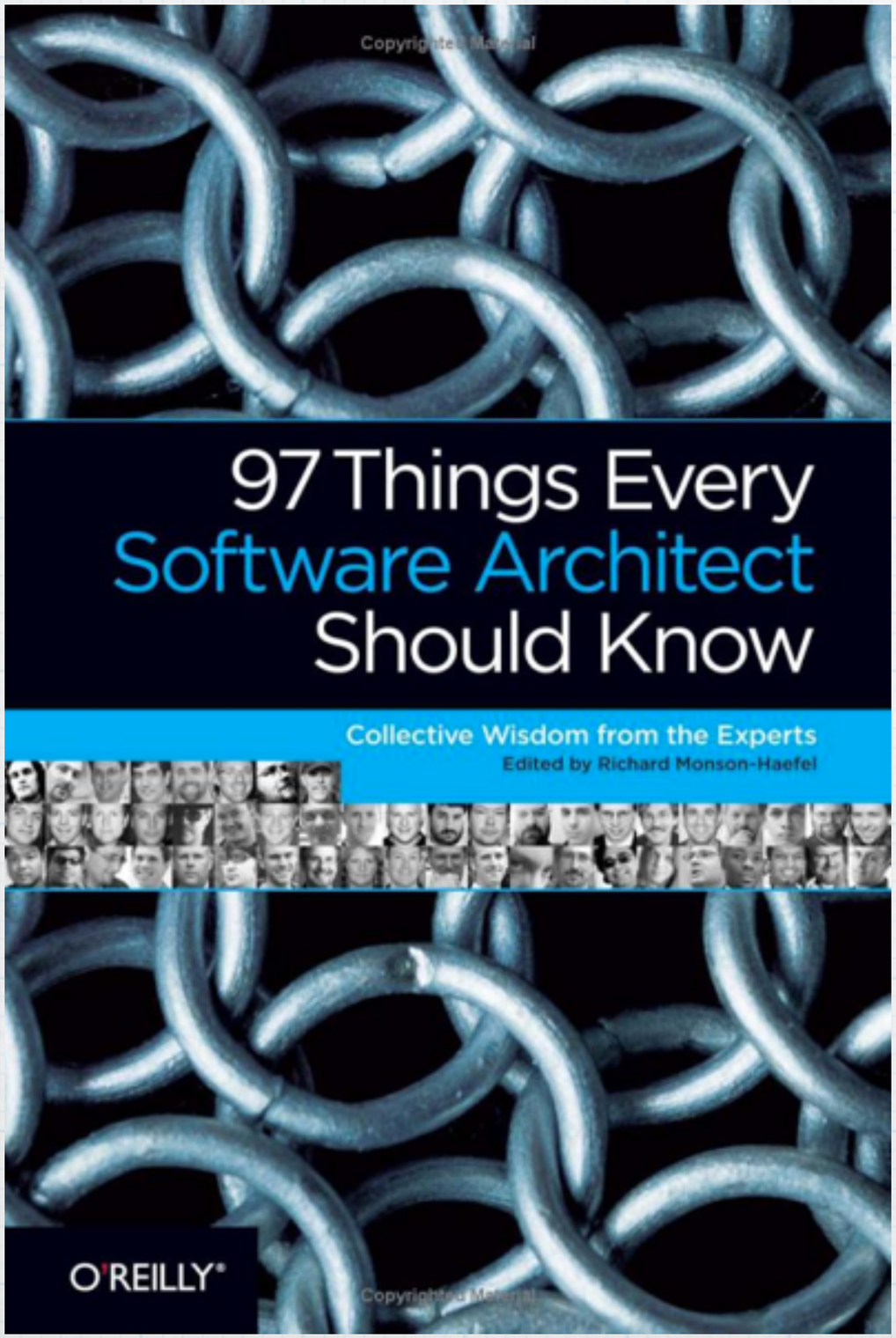
"Burk fixed more code than we care to admit."
— Kathy Sierra & Bert Bates,
Authors: SCJP6 Study Guide

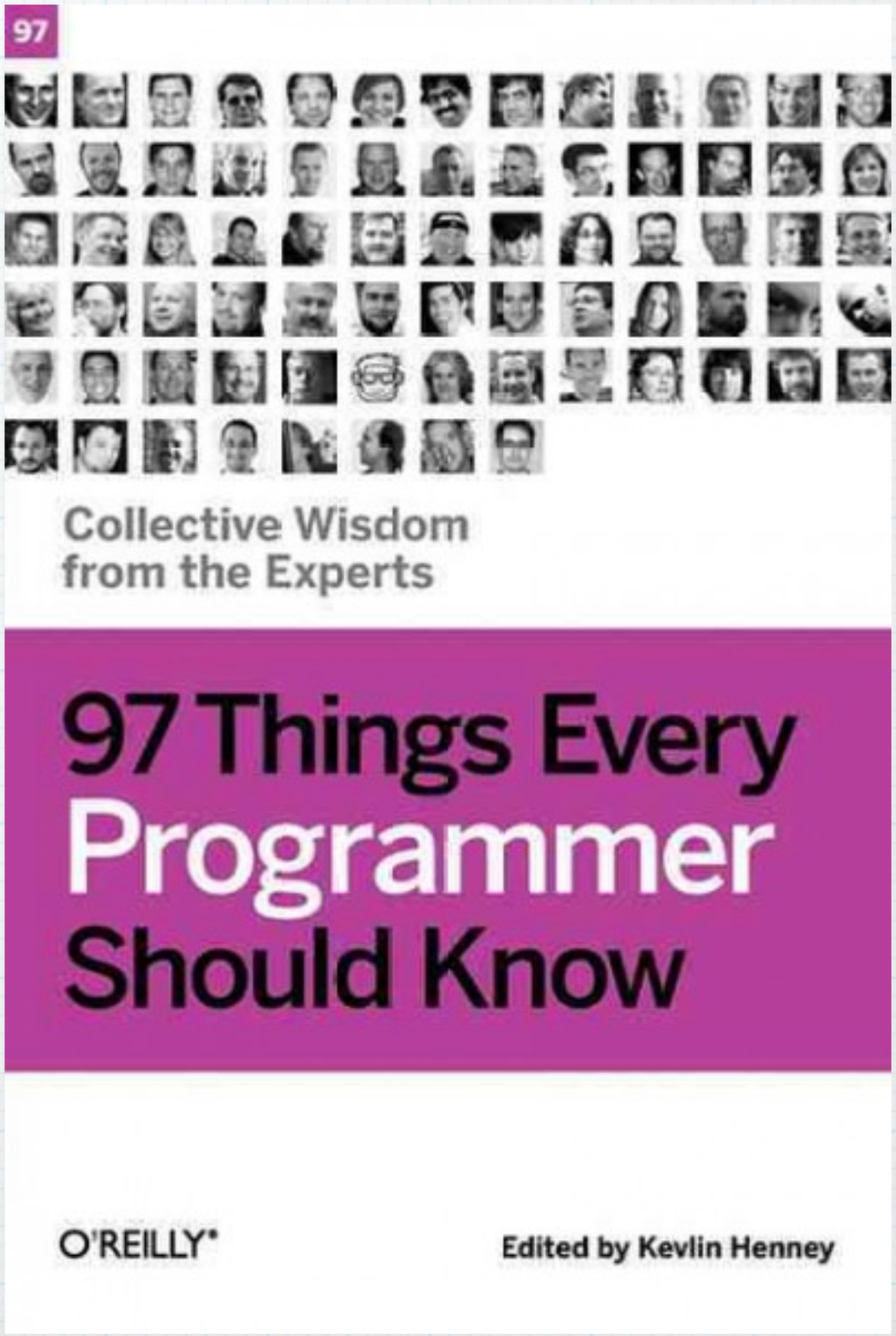


Sun Certified Java Programmer,
Developer, Enterprise Architect

Programmer and Architect with
Daugherty Business Solutions

Who is Burk Hufnagel?

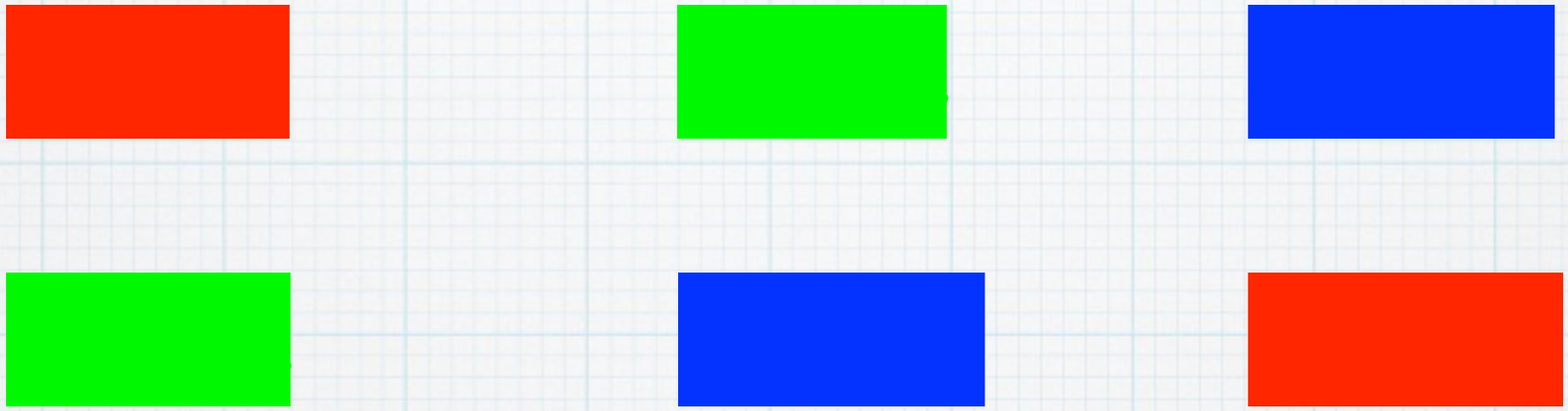




Who are you?

- Java developer and/or architect
- Interested in learning to deliver better code faster
- Willing to try something new

Color Fun



Session Goal

You will know how to Deliver Better Code, Faster, with Spock

Software Development Life Cycle

Read requirements

Analysis - Decide what to write

Development - write, test, and debug code

Turn over to QA

Find bugs

Turn over to UAT

Find bugs

Release to production

Find bugs

Pop Quiz!

Please clap for your development process:

* No automated tests for production code

* Automated tests are written after production code

* Automated tests are written before production code

Wouldn't it be dreamy if...

stock photo

Wouldn't it be dreamy if...

* You had accurate and clearly understood business requirements for your projects
* You regularly delivered new features, backed by clean code, faster than you do now
* Few, if any, bugs make it to QA
* It's fast & easy to find & fix any that do
* Your developer docs were always up to date
* You don't spend any time updating the docs

* You could have it all just by writing some code...

Session Goal

You will know how to Deliver Better Code, Faster, with Spock

Session Goal

Deliver

Better Code

Faster

Spock

Deliver

* Deliver means code is in production
  * Until then there is no business value

* Time to deliver includes:
  * Time to write, test, and debug code
  * Time spent in QA
  * Time spend in UAT

Session Goal

Deliver

"Finished coding" != "Ready to deploy"

Better Code

Faster

Spock

Better Code

* Build the right thing
* Build the thing right

Build the
right thing

Build the
thing right

Session Goal

Deliver

"Finished coding" != "Ready to deploy"

Better Code

Meets requirements and high quality

Faster

Spock

Faster

- Faster doesn't mean quickly cranking out production code
- Faster is about getting code designed, written, debugged, tested, and into production over and over.
- If the team can't maintain the pace, the team must slow down

Session Goal

Deliver

"Finished coding" != "Ready to deploy"

Better Code

Meets requirements and high quality

Faster

Less time than your current process

Spock

Spock

Spock

- BDD-Style framework designed for developers. It supports writing unit, integration, and user acceptance tests

- Written in Groovy, a JVM-based language designed to make life easier for Java developers

- Like its namesake, Spock benefits from the heritage of two worlds; collaboration tools like Cucumber, and testing tools like JUnit.

Session Goal

Deliver

"Finished coding" != "Ready to deploy"

Better Code

Meets requirements and high quality

Faster

Less time than your current process

Spock

Groovy-based BDD framework for developers

The Process - BDD Light

* Specification by Example
* Test Driven Development using higher level specifications to drive lower level specifications
* Generate "living documentation", from the results of executing the specifications

TDD Basics

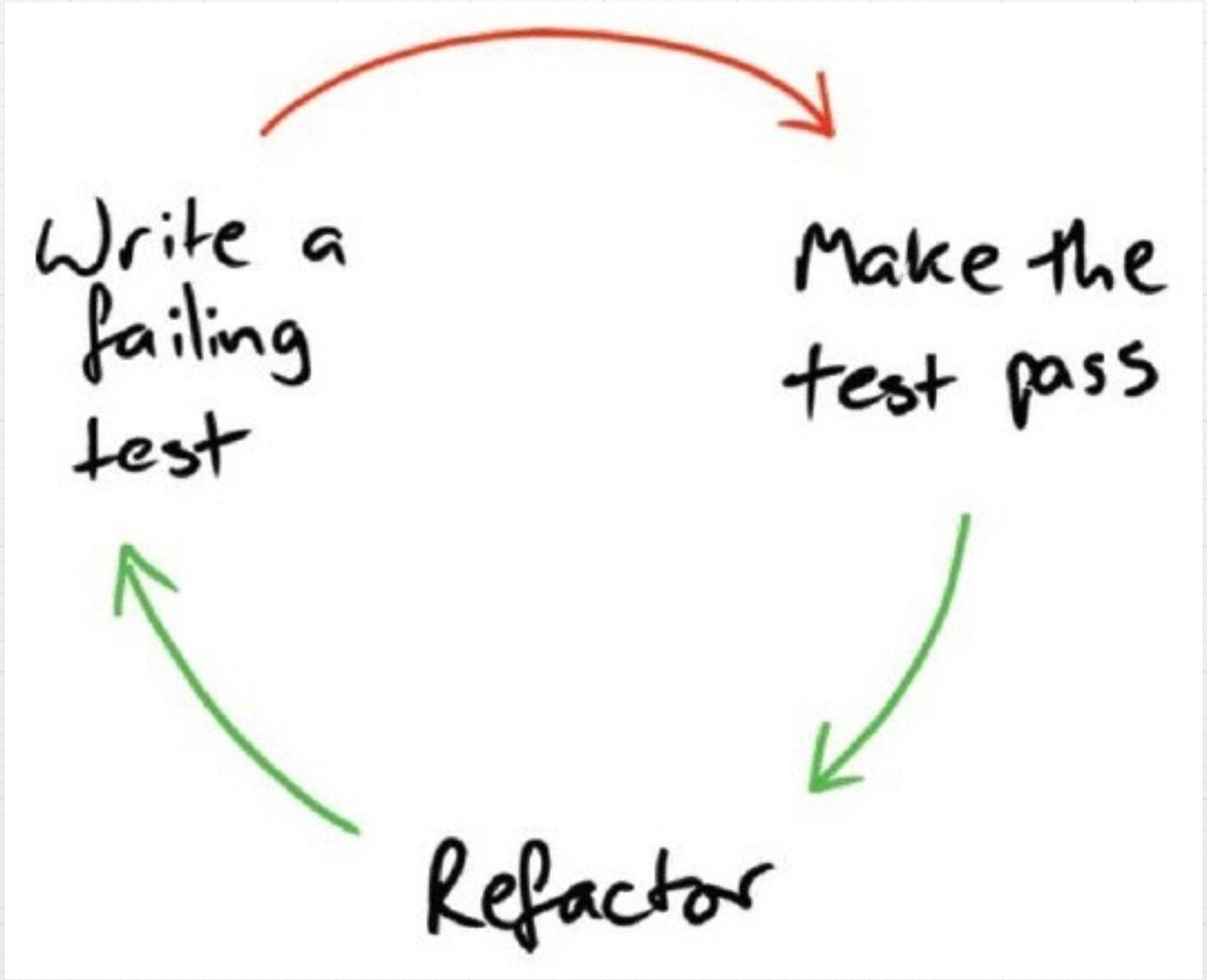


image from: a4direct.com

TDD Adoption Issues

- The promise of TDD is seems illogical. How can adding time writing test code reduce the overall time?

Thought Experiment

image from: ottowa-snapshots.net

Thought Experiment

Take aways:

* Starting the work before before we understand exactly what is needed takes longer and results in wasted time and materials.

* By investing time up front to we understand exactly what is needed, delivering the right solution takes less overall time.

TDD Adoption Issues

- TDD is seems illogical. How can adding time writing test code reduce the overall time?
- Changing how you do something causes a temporary drop in productivity
- "Our code it too complex to test"

Behavior Driven Development

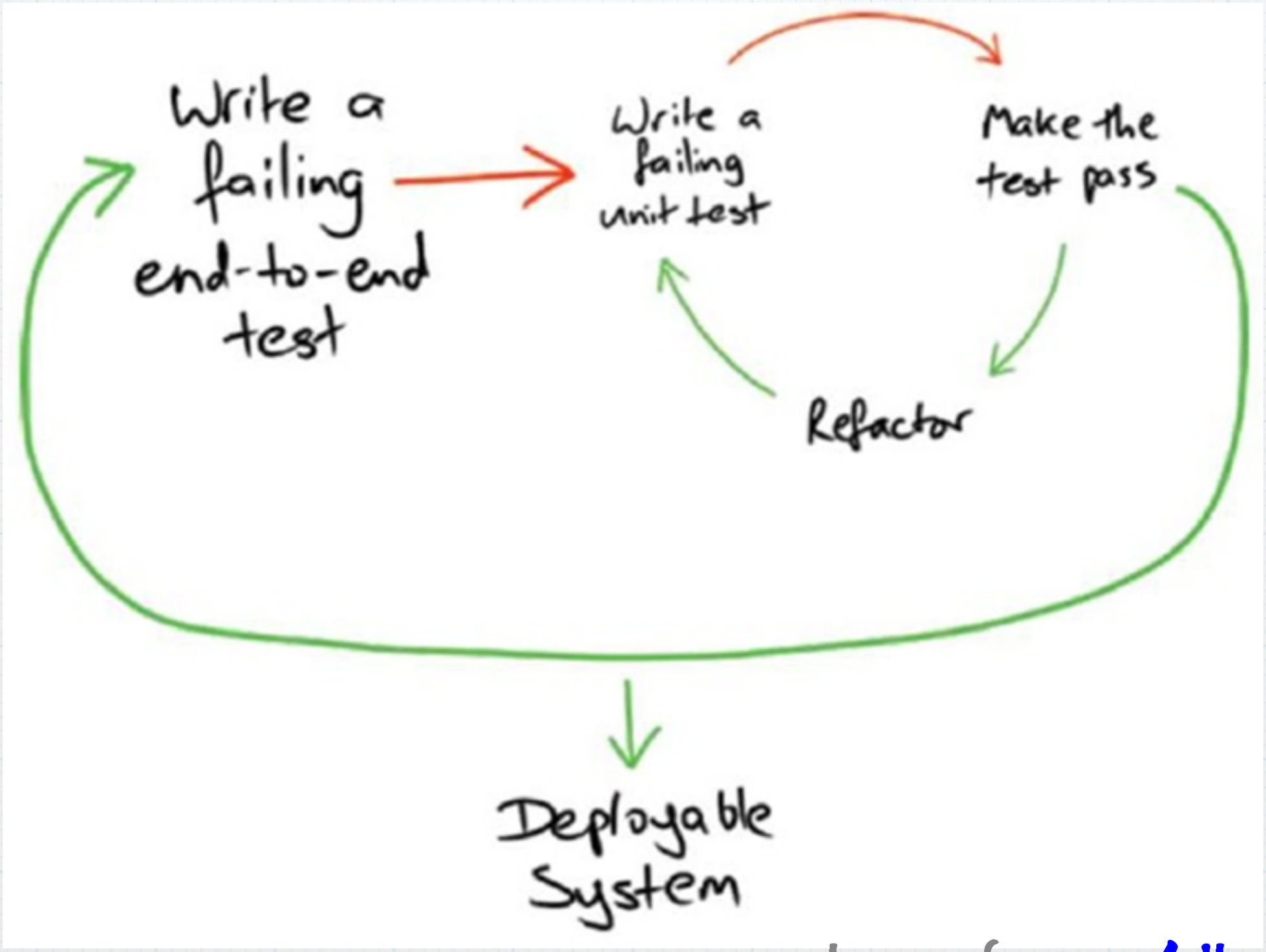
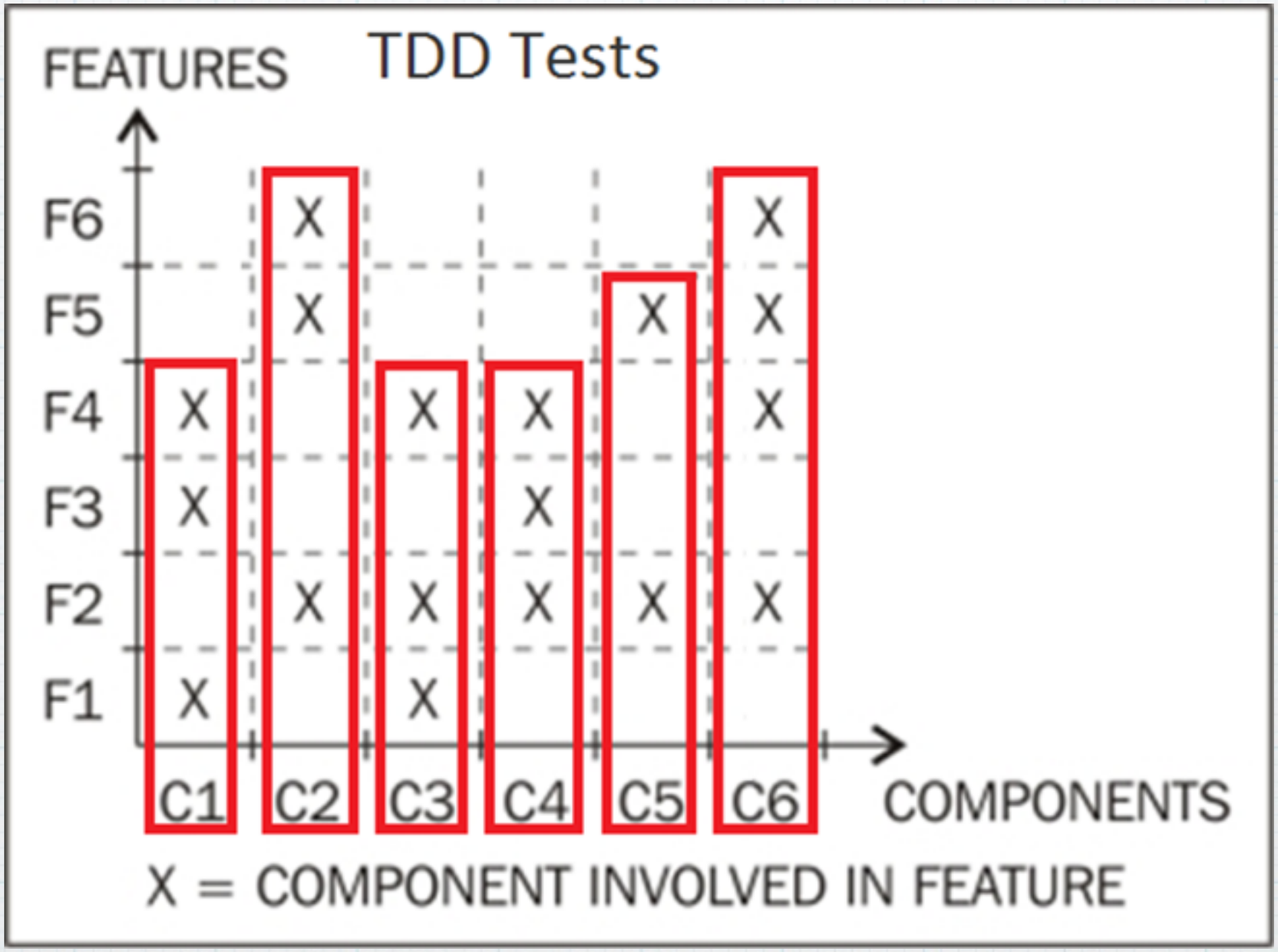


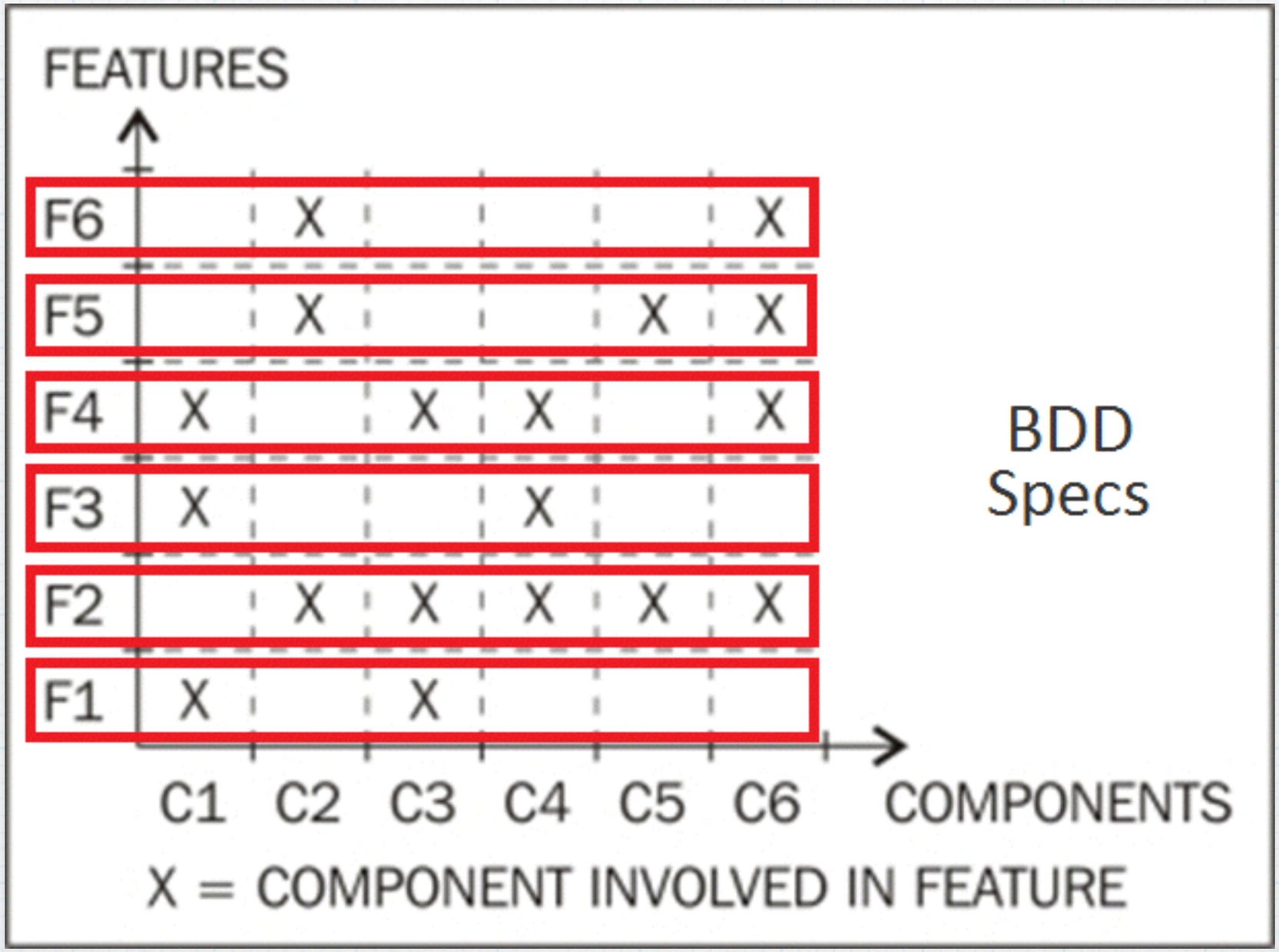
image from: a4direct.com

TDD vs BDD



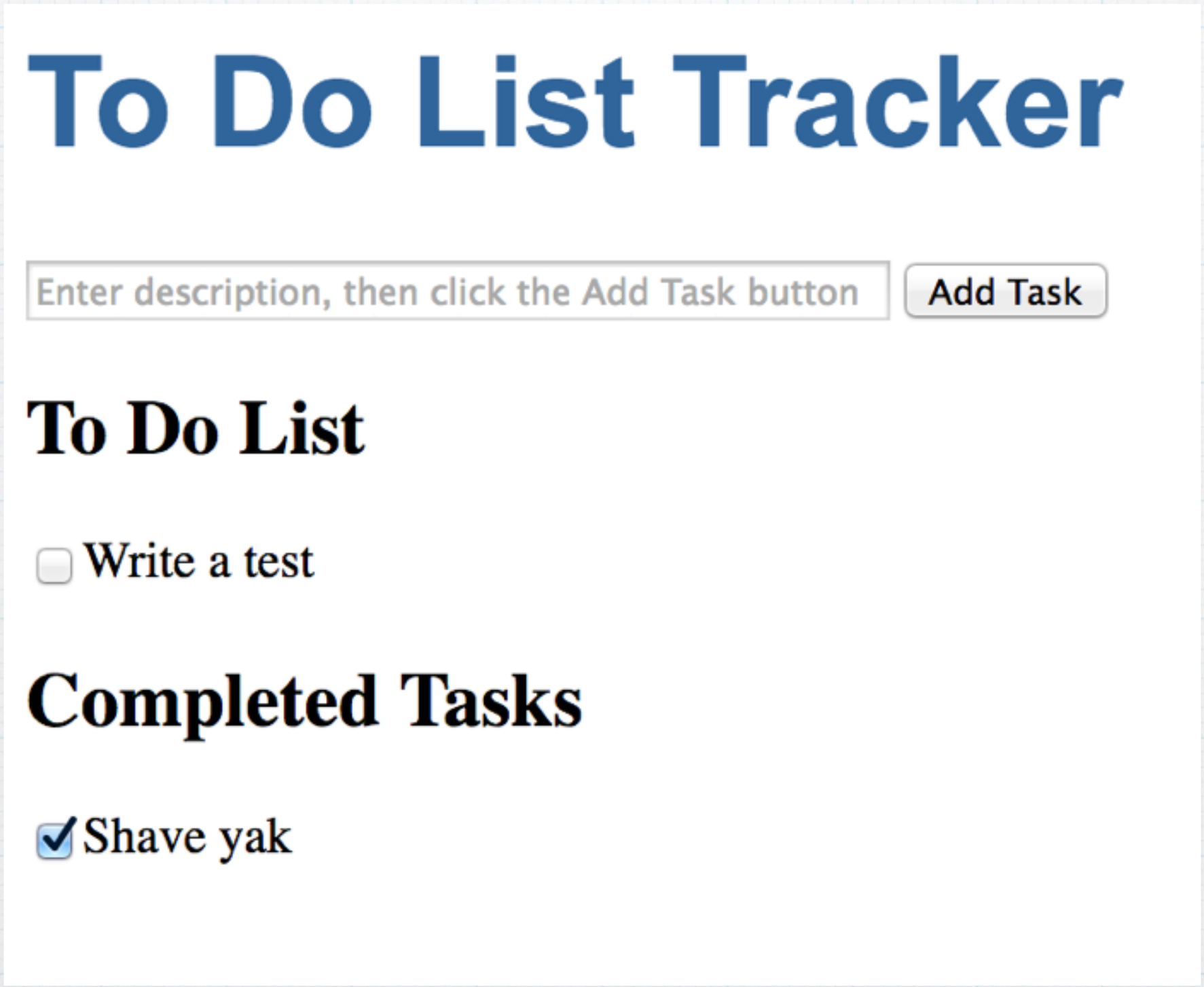
Modified diagram from “Learning Behaviour-driven Development with JavaScript”

TDD vs BDD



Modified diagram from “Learning Behaviour-driven Development with JavaScript”

Let's see it in action...



https://cucumber.io

Docs Blog Events School Training Cucumber Pro Support

cucumber

Simple, human collaboration

1. Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

2. Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

3. Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

6. Repeat 2-5 until green like a cuke

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/step_d
    Then the result should be 120 on the screen      # features/step_d
```

7. Repeat 1-6 until the money runs out

Cucumber lets software development teams

Cucumber works with Ruby, Java, .NET, Flex or

Cucumber also supports more succinct tests in

Cucumber

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

Scenario: Adding an item to an empty list
Given an empty list
When I add "Shave yak"
Then it should be displayed in the To Do area

Cucumber

```
@Given("an empty list")
public void create_an_empty_list() {
    createEmptyList();
}
@When("I add \"([^\"]*)\"")
public void add_task(String text) {
    theWorld.taskDescription = text;
    submitTask( taskDescription );
}
@Then("the To Do area should contain the task")
public void todo_should_contain_task() {
   assertTrue(toDoArea.contains(theWorld.taskDescription));
}
```

Spock

```
@Title("Tracking items with my To Do List")
@Narrative('''As a human with too much on my mind
I want a To Do List to track tasks
So that I don't have to remember them all''')
class ToDoListAppSpec extends Specification {
  def "Adding a task to an empty list"() {
    given: "an empty list"
      createAnEmptyList();
    when: 'I add "Shave yak"'
      def taskDescription = 'Shave yak'
      submitTask( taskDescription );
    then: "the To Do area should contain the task"
      toDoAreaContains( taskDescription ) == true;
  }
}
```

Spock Reports

Report for com.burkhufnagel.TodolistApplicationSpec

Summary:

Created on Mon Oct 02 23:19:07 PDT 2017 by bth0624

Executed features	Failures	Errors	Skipped	Success rate	Time
3	0	0	0	100.0%	0.023 seconds

Tracking items with my To Do List <—@Title

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

<—@Narrative

Features:

- Adding a task to an empty list
- Completing a task in my list
- Removing a task from my list

Adding a task to an empty list <—@Method name

Return

Given: an empty list
When: I add "Shave yak"
Then: the To Do area should contain the task

<—@Given, When, Then

Completing a task in my list

Return

Given: a list containing the task "Shave yak"
When: I mark "Shave yak" as completed

Summary:

Created on Mon Oct 02 23:28:42 PDT 2017 by bth0624

Executed features	Failures	Errors	Skipped	Success rate	Time
3	1	0	0	66.67%	0.069 seconds

Tracking items with my To Do List

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

Features:

- Adding a task to an empty list
- Completing a task in my list
- Removing a task from my list

Adding a task to an empty list

Return

Given: an empty list
When: I add "Shave yak"
Then: the To Do area should contain the task

The following problems occurred:

- `Condition not satisfied:`

```
tasks.contains( taskDescription )
|      |            |
|      false        Shave yak
[Make this work, Learn BDD]
```

<— failing line of code
<— why it failed

Behavior Driven Development

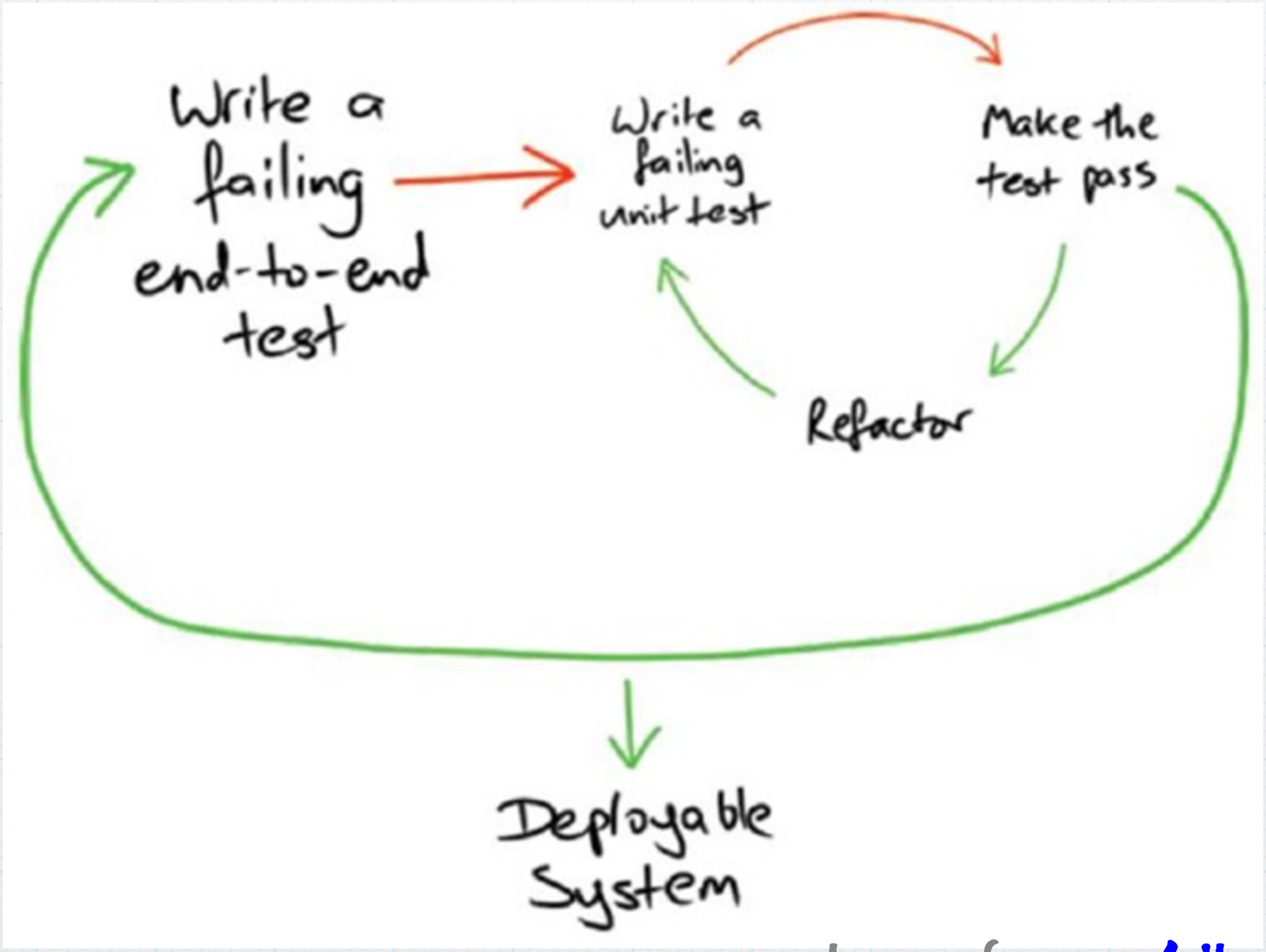


image from: a4direct.com

A scenario describes high level behavior:

Scenario: Adding an item to an empty list

Given an empty list

When I add "Shave yak"

Then it should be displayed in the To Do area

The Spock specification looks like this...

```
def "Adding a task to an empty list"() {
   given: "an empty list"
      createAnEmptyList()

   when: 'add "Shave yak"'
      def taskDescription = 'Shave yak'
      submitTask( taskDescription )

   then: "the To Do area should contain the task"
      toDoAreaShouldContain( taskDescription ) == true
}
```

which should suggests integration and unit tests.

TDD in action

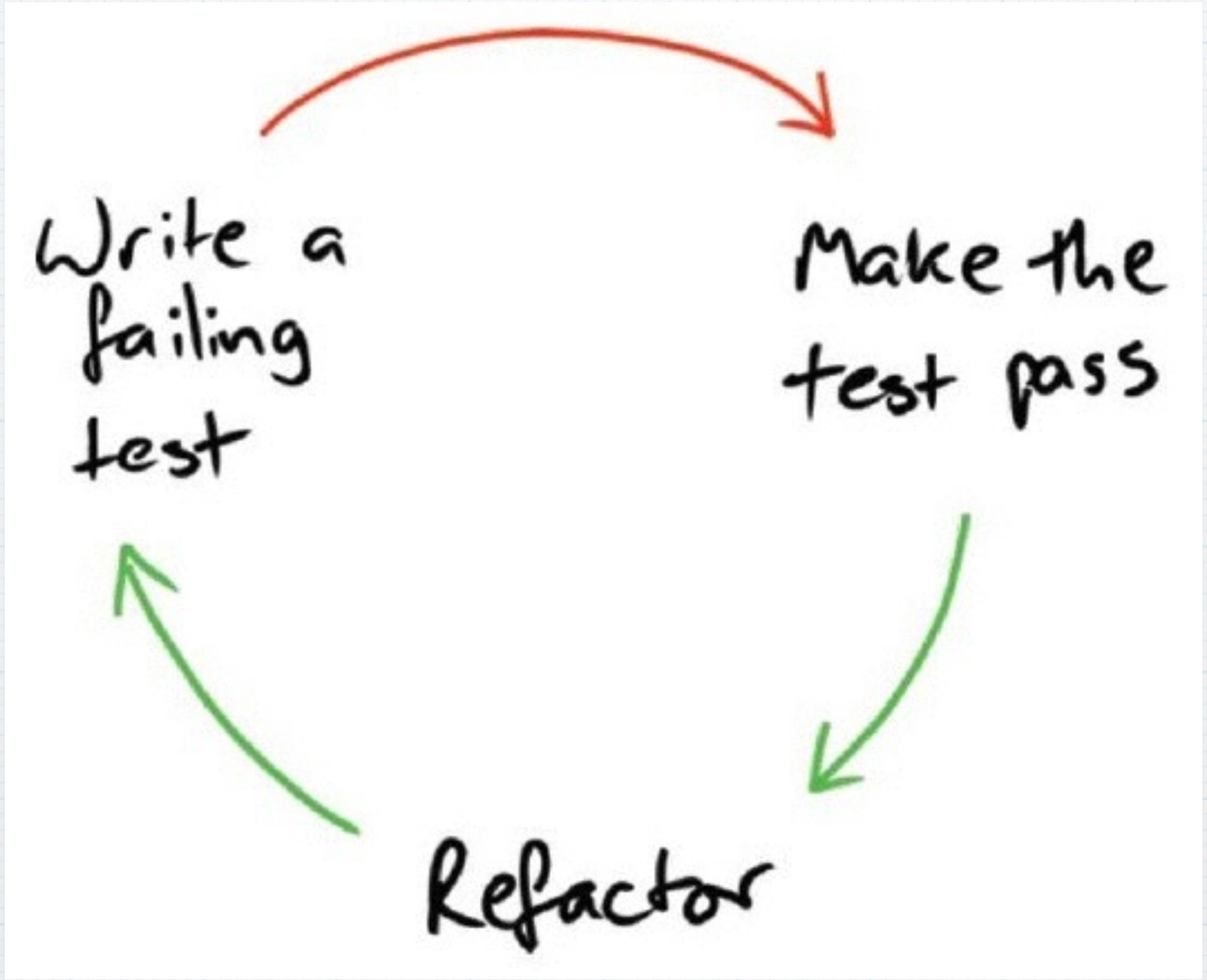


image from: a4direct.com

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.

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"

      when: "a task is added to the list"

      then: "the list should contain the task"

   }
}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"

      then: "the list should contain the task"

   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"

      then: "the list should contain the task"

   }
}
```

```groovy
public class ListTracker {

}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"

      then: "the list should contain the task"

   }
}
```

```
public class ListTracker {

}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"

   }
}
```

```
public class ListTracker {

}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"

   }
}
```

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

   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"

   }
}
```

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

}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks();
   }
}
```

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

   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
   }
}
```

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

   }

   public List<String> getTasks() {

   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
   }
}
```

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

   }

   public List<String> getTasks() {

   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
   }
}
```

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

   }

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

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
   }
}
```

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

   }

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

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   }

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

```groovy
class ListTrackerSpecification extends Specification {
    def "When a task is added to the list, it should be retrievable"() {
        given: "an empty task list"
            def listTracker = new ListTracker()
        when: "a task is added to the list"
            listTracker.addTask( "Shave Yak" )
        then: "the list should contain the task"
            def tasks = listTracker.getTasks()
            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

```groovy
    }

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

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   }

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

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   }

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

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   }

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

```groovy
class ListTrackerSpecification extends Specification {
    def "When a task is added to the list, it should be retrievable"() {
        given: "an empty task list"
            def listTracker = new ListTracker()
        when: "a task is added to the list"
            listTracker.addTask( "Shave Yak" )
        then: "the list should contain the task"
            def tasks = listTracker.getTasks()
            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

```groovy
    }

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

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   }

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

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

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

   public void addTask( String taskDescription ) {

   }

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

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak” )
   }
}
```

```groovy
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;
   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         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

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

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

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         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;
   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yak" )
   }
}
```

```groovy
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;
   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yaks" )
   }
}
```

```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;
   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
    def "When a task is added to the list, it should be retrievable"() {
        given: "an empty task list"
            def listTracker = new ListTracker()
        when: "a task is added to the list"
            listTracker.addTask( "Shave Yak" )
        then: "the list should contain the task"
            def tasks = listTracker.getTasks()
            tasks.contains( "Shave Yaks" )
    }
}
```

Console | Markers | Progress | JUnit

Finished after 0.334 seconds

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

com.burkhufnagel.
When a task is

Failure Trace

Condition not satisfied:

```
tasks.contains("Shave Yaks")
|     |
|     false
[Shave Yak]
```

at com.burkhufnagel.ListTrackerSpecification.When a task is to the list, it should be retrievable(ListTrackerSpecification.groovy:40)

```
        return tasks;
    }
}
```

```groovy
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list”
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yaks” )
   }
}
```

```groovy
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;
   }
}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list”
         def taskDescription = "Shave Yak"
         listTracker.addTask( "Shave Yak" )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         tasks.contains( "Shave Yaks” )
   }
}
```

```
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;
   }
}
```

```
class ListTrackerSpecification extends Specification {
   def "When a task is added to the list, it should be retrievable"() {
      given: "an empty task list"
         def listTracker = new ListTracker()
      when: "a task is added to the list”
         def taskDescription = "Shave Yak"
         listTracker.addTask( taskDescription )
      then: "the list should contain the task"
         def tasks = listTracker.getTasks()
         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;
   }
}
```

```groovy
class ListTrackerSpecification extends Specification {
    def "When a task is added to the list, it should be retrievable"() {
        given: "an empty task list"
            def listTracker = new ListTracker()
        when: "a task is added to the list”
            def taskDescription = "Shave Yak"
            listTracker.addTask( taskDescription )
        then: "the list should contain the task"
            def tasks = listTracker.getTasks()
            tasks.contains( taskDescription )
    }
}
```

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

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

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

Sample Spock Report

Report for com.burkhufnagel.ListTrackerSpecification

Summary:

Created on Mon Sep 25 00:36:16 EDT 2017 by bth0624

Executed features	Failures	Errors	Skipped	Success rate	Time
1	0	0	0	100.0%	0.019 seconds

Features:

- When a task is added to the list, it should be retrievable

When a task is added to the list, it should be retrievable

Return

Given: an empty task list

When: a task is added to the list

Then: the list should contain the task

Why Spock?

Spock vs traditional Java tools

UAT:	Spock	Cucumber
HTTP:	Geb	Selenium
Unit Test:	Spock	JUNIT
Mock/Spy:	Spock	Mockito
		PowerMock

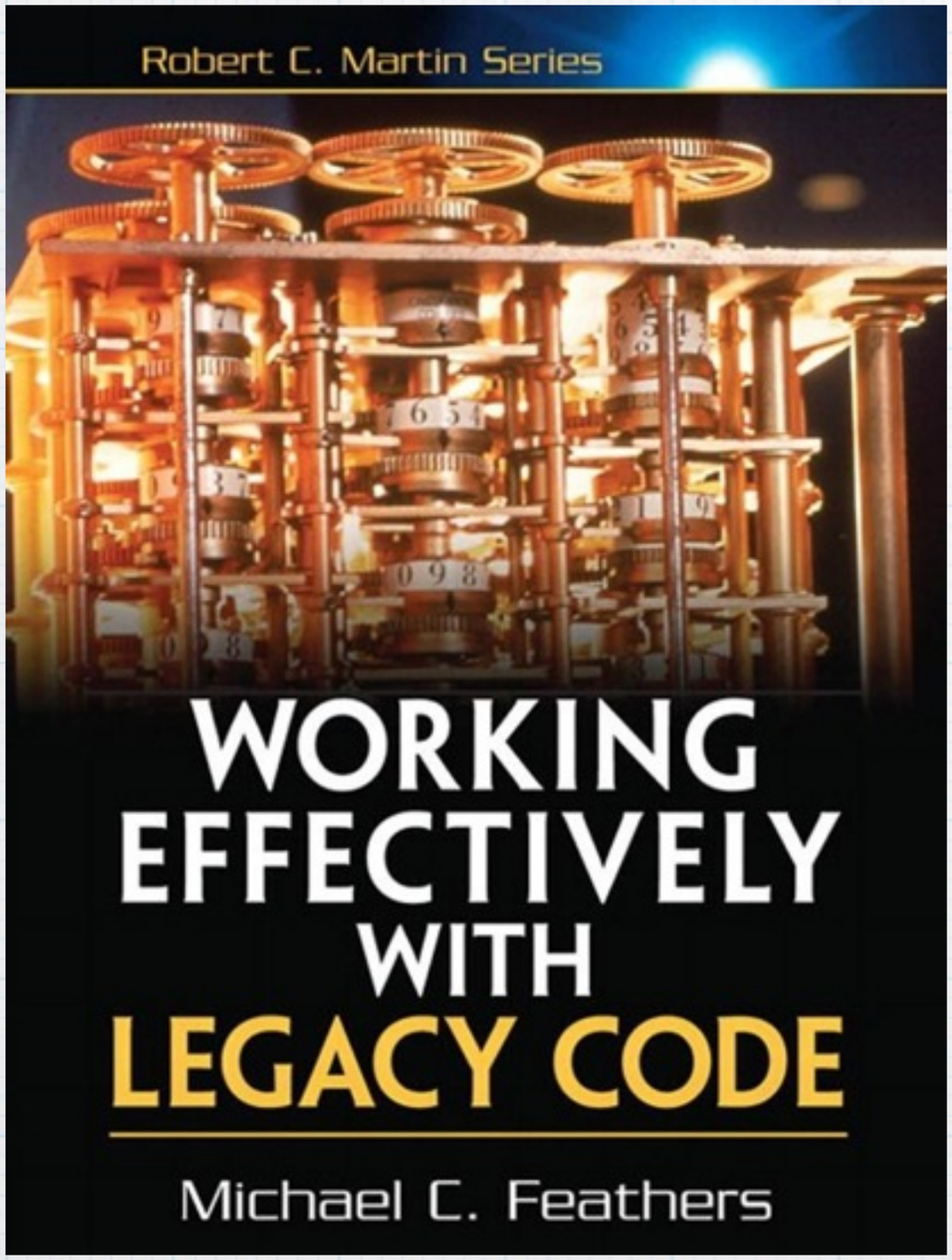
Session Goal

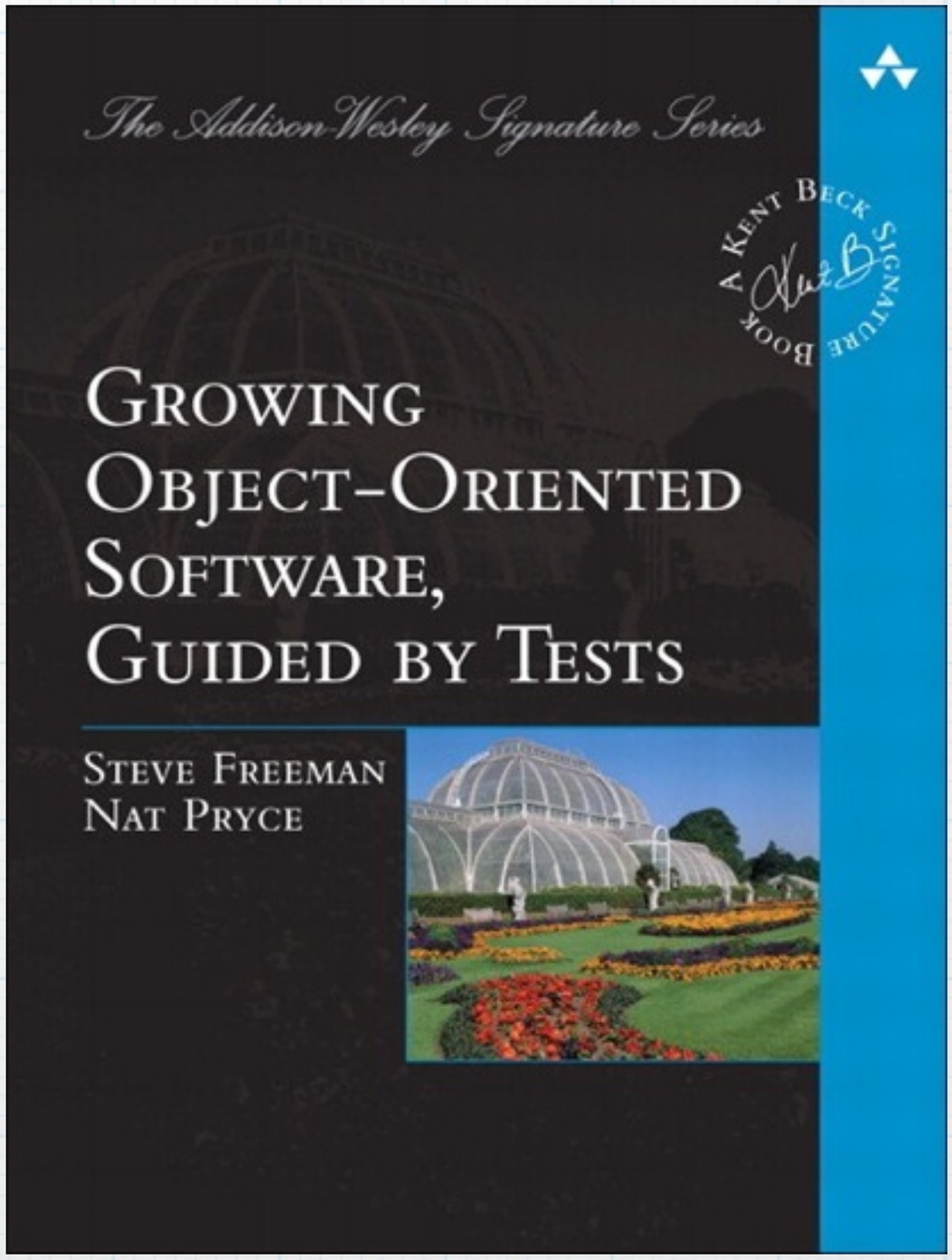
You will know how to
Deliver Better Code, Faster,
with Spock

Thank You!

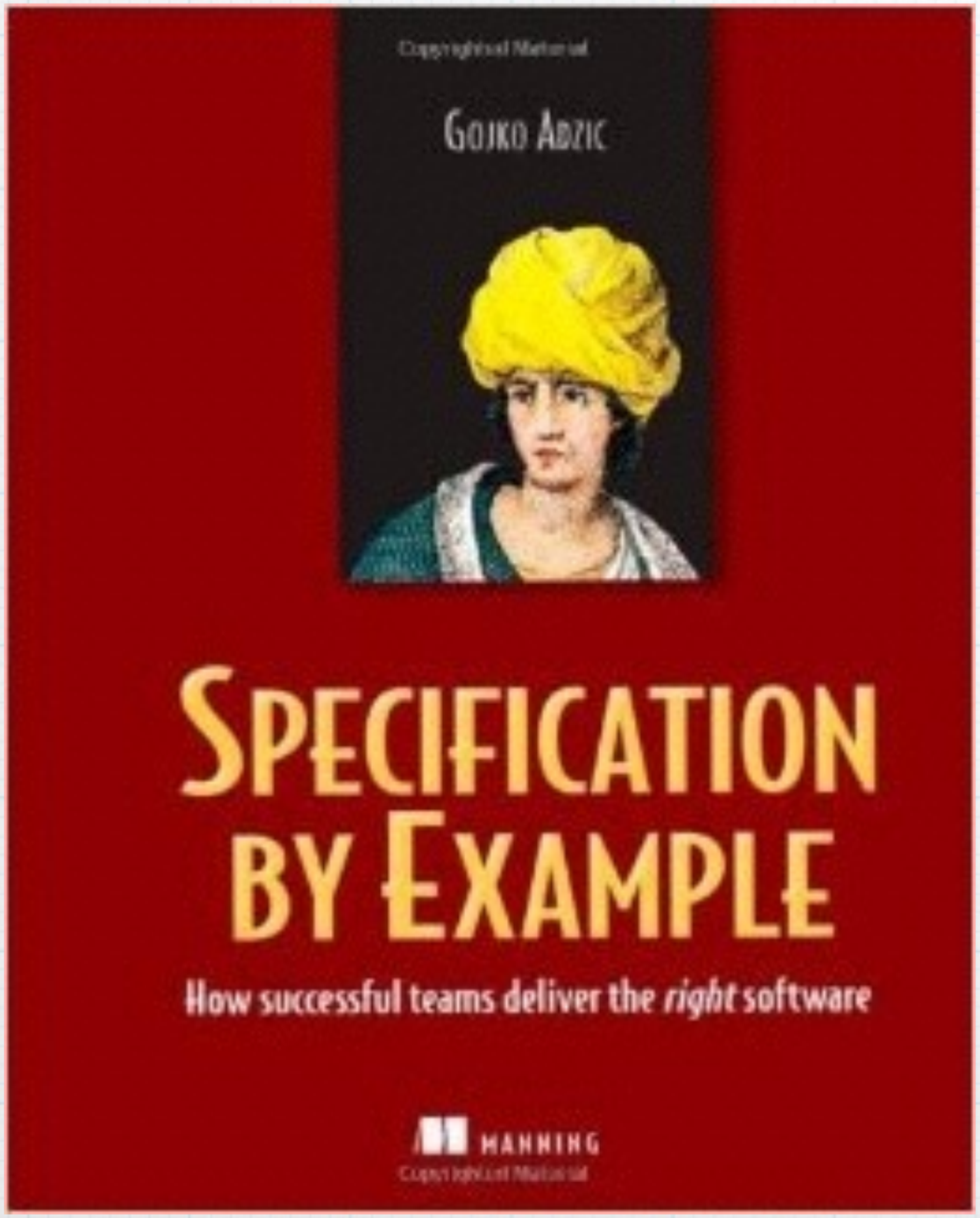
Test well, and Prosper.

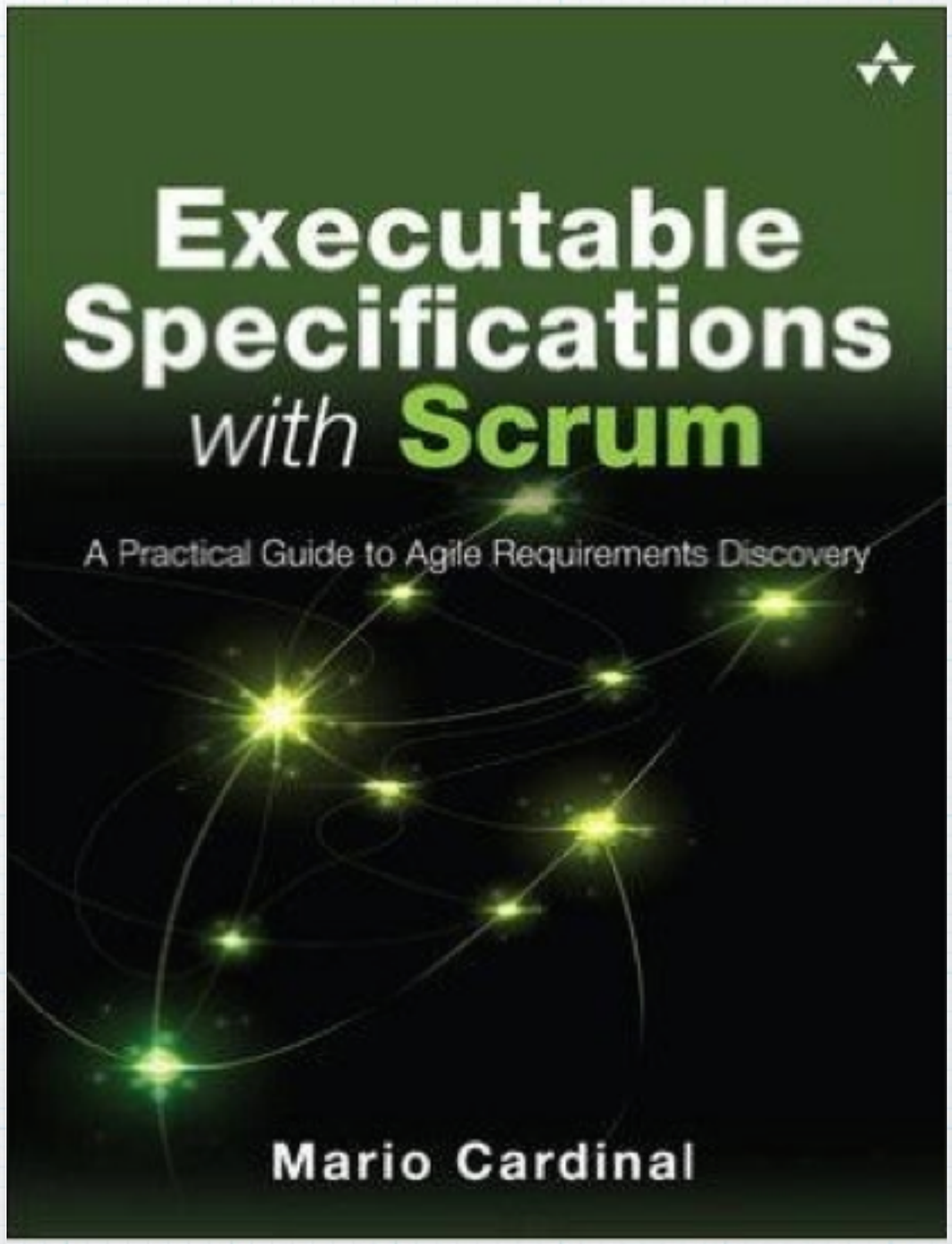
Recommended Reading





Recommended Reading





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