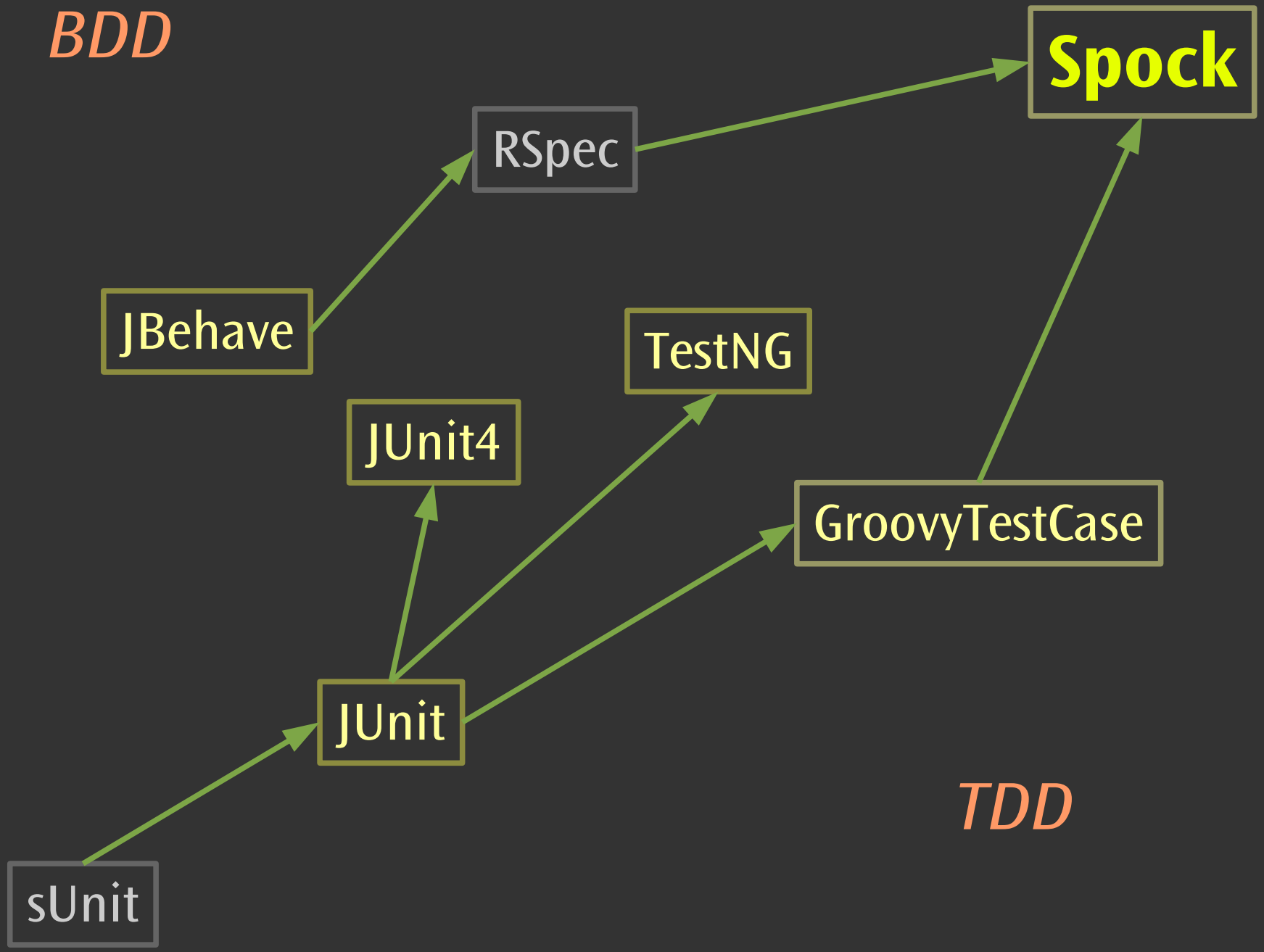
Spocktacular Testing

Russel Winder

email: russel@winder.org.uk
twitter: @russel_winder
Web: http://www.russel.org.uk

Opening

An Historical Perspective



Spock is Groovy-based...

...but can test any JVM-based code.

NB Testing frameworks support integration and system testing as well as unit testing.

Testing

- Unit:
  - Test the classes, functions and methods to ensure they do what we need them to.
  - As lightweight and fast as possible.
  - Run all tests always.
- Integration and system:
  - Test combinations or the whole thing to make sure the functionality is as required.
  - Separate process to create a "sandbox".
  - If cannot run all tests always, create smoke tests.

Code Under Test

```
static message() {
  'Hello World.'
}

println message()
```

helloWorld.groovy

Unit Test

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

import helloWorld

class Test_HelloWorld extends Specification {
  def 'ensure the message function returns hello world'() {
    expect:
     helloWorld.message() == 'Hello World.'
  }
}
```

System Test

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class Test_HelloWorld extends Specification {
  def 'executing the script results in hello world on the standard output'() {
    given:
     def process = 'helloWorld.groovy'.execute()
    expect:
     process.waitFor() == 0
     process.in.text == 'Hello World.\n'
  }
}
```

A bit less Groovy...

Code under Test

```
package uk.org.winder.spockworkshop;

class HelloWorld {
  private static String message() {
    return "Hello World.";
  }
  public static void main(final String[] args) {
    System.out.println(message());
  }
}
```

HelloWorld.java

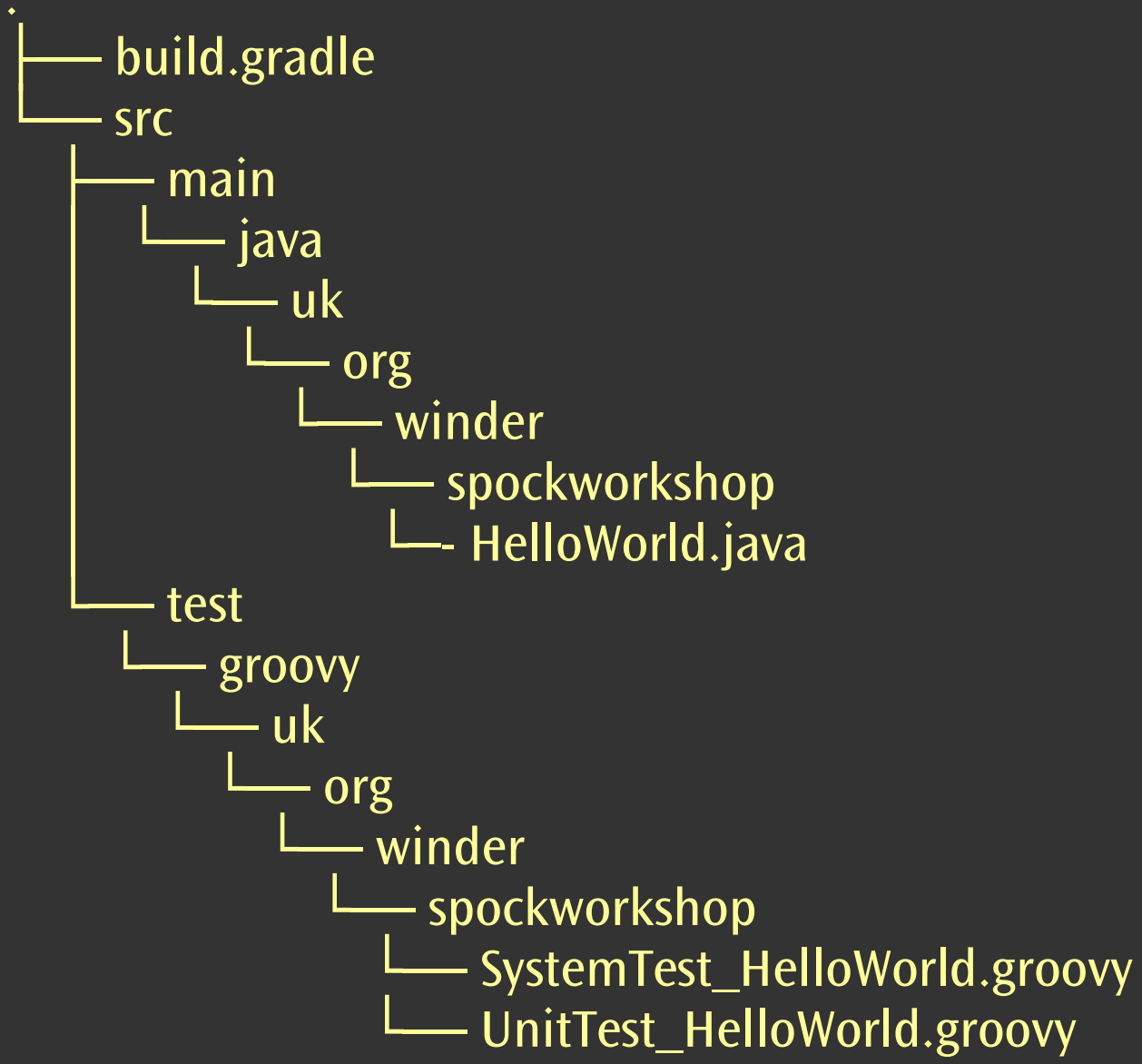
Unit Test

```
package uk.org.winder.spockworkshop

import spock.lang.Specification

class UnitTest_HelloWorld extends Specification {
  def 'ensure the message function returns hello world'() {
    expect:
     HelloWorld.message() == 'Hello World.'
  }
}
```

System Test

```
package uk.org.winder.spockworkshop

import spock.lang.Specification

class SystemTest_HelloWorld extends Specification {
  def 'executing the program results in hello world on the standard output'() {
    given:
     def process = ['java', '-cp', 'build/classes/main',
        'uk.org.winder.spockworkshop.HelloWorld'].execute()
    expect:
     process.waitFor() == 0
     process.in.text == 'Hello World.\n'
  }
}
```

Project Structure



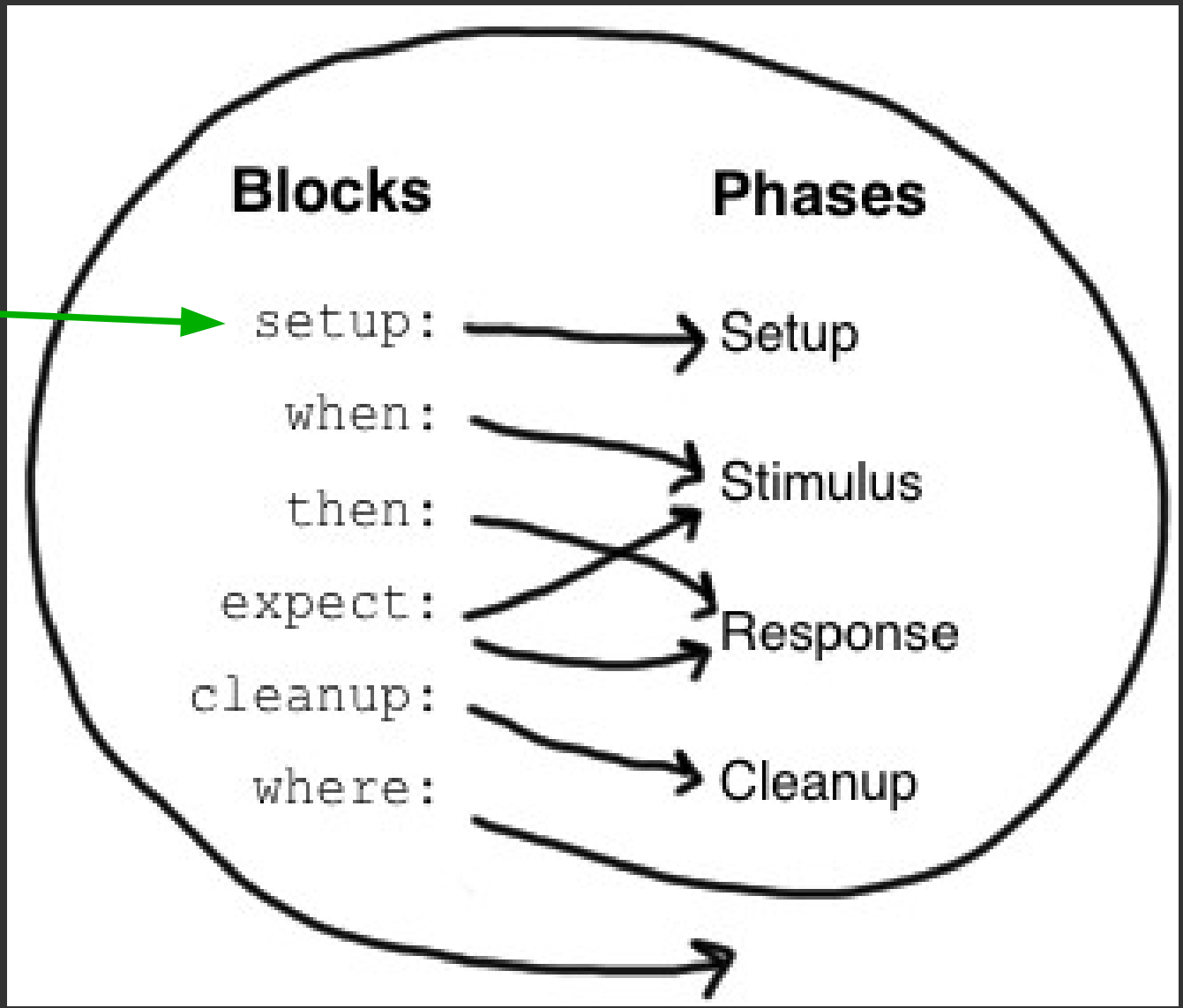
Build — Gradle

```
apply plugin: 'groovy'
apply plugin: 'application'

repositories {
  jcenter()
  mavenCentral()
}

dependencies {
  testCompile 'org.spockframework:spock-core:1.0-groovy-2.4'
}

mainClassName = 'uk.org.winder.spockworkshop.HelloWorld'
```

Moving On

Spock Test Structure

given:



Code Under Test

```
static message() {
  'Hello World.'
}

println message()
```

helloWorld.groovy

Unit Test

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

import helloWorld

class Test_HelloWorld extends Specification {
  def 'ensure the message function returns hello world'() {
    expect:
     helloWorld.message() == 'Hello World.'
  }
}
```

System Test

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class Test_HelloWorld extends Specification {
  def 'executing the script results in hello world on the standard output'() {
    given:
     def process = 'helloWorld.groovy'.execute()
    expect:
     process.waitFor() == 0
     process.in.text == 'Hello World.\n'
  }
}
```

Another Code Under Test

```
class Stuff {
  private final data = []
  def leftShift(datum) {
    data << datum
  }
}
```

Stuff.groovy

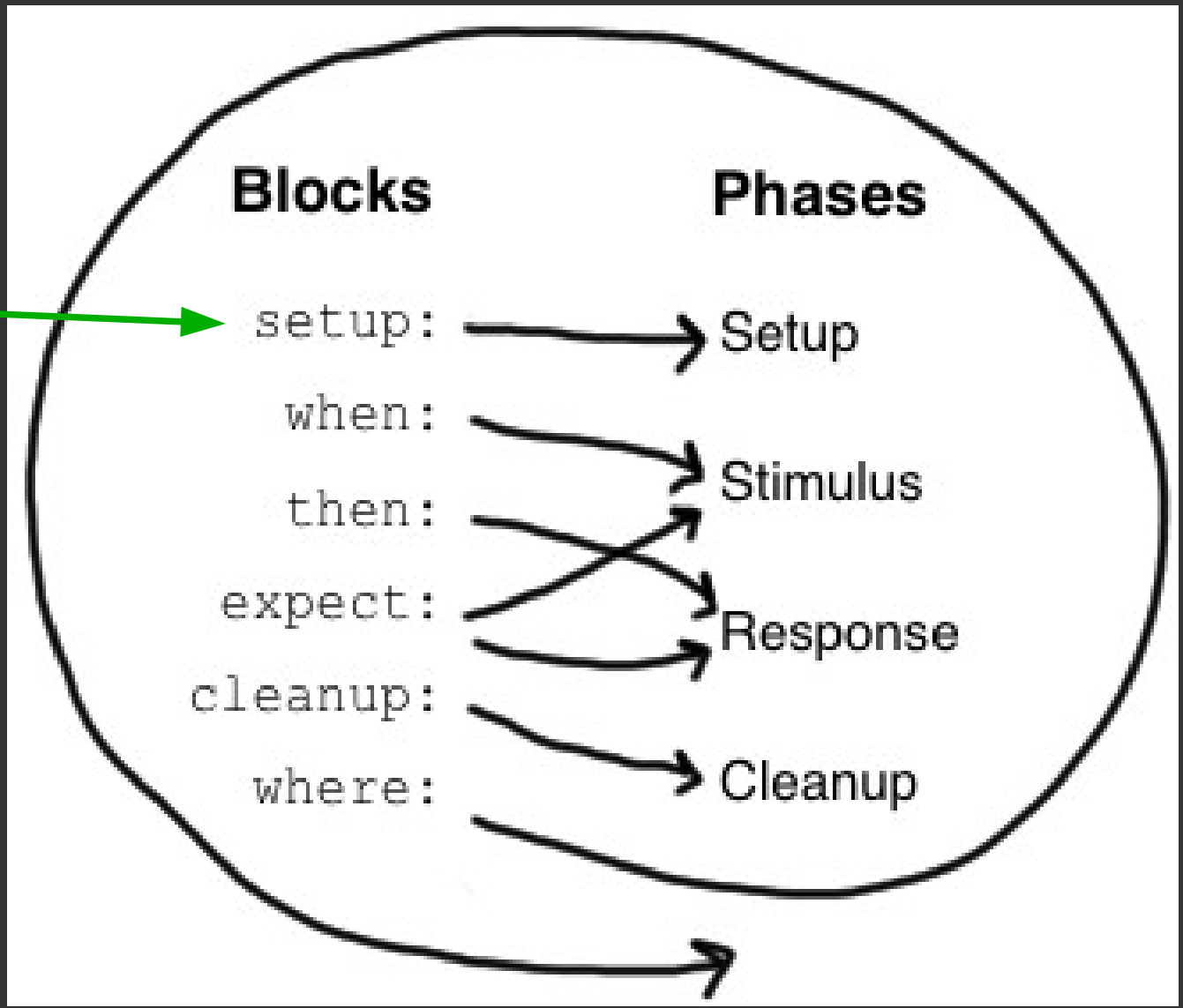
Unit Testing It

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

import Stuff

class TestStuff extends Specification {
  def 'check stuff'() {
    given:
    def stuff = new Stuff()
    expect:
    stuff.data == []
    when:
    stuff << 6
    then:
    stuff.data == [6]
    when:
    stuff << 6
    then:
    stuff.data == [6, 6]
  }
```

```
  def 'check other stuff'() {
    given:
    def stuff = new Stuff()
    expect:
    stuff.data == []
    when:
    stuff.leftShift(6)
    then:
    stuff.data == [6]
    when:
    stuff.leftShift(6)
    then:
    stuff.data == [6, 6]
  }
}
```

Data-driven Testing

given:



Code Under Test

```
class Id {
  def eval(x) { x }
}
```

Id.groovy

Unit Test Code

```
#! /usr/bin/env groovy

@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification
import spock.lang.Unroll

class idTest extends Specification {
  @Unroll def 'id.eval always returns the value of the parameter: #i'() {
    given:
    final id = new Id ()
    expect:
    id.eval(i) == i
    where:
    i << [0, 1, 2, 3, 's', 'ffff', 2.05]
  }
}
```

Code Under Test

```
class Functions {
  static square(x) { x * x }
}
```

Functions.groovy

Unit Test — Variant 1

```
#! /usr/bin/env groovy

@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification
import spock.lang.Unroll

class functionsTest_alt_1 extends Specification {
  @Unroll def 'square always returns the square of the parameter: #x'() {
    expect:
    Functions.square(x) == r
    where:
    x << [0, 1, 2, 3, 1.5]
    r << [0, 1, 4, 9, 2.25]
  }
}
```

Unit Test — Variant 2

```
#! /usr/bin/env groovy

@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification
import spock.lang.Unroll

class functionsTest_alt_2 extends Specification {
  @Unroll def 'square always returns the square of the parameter: #x'() {
    expect:
    Functions.square(x) == r
    where:
    [x, r] << [[0, 0], [1, 1], [2, 4], [3, 9], [1.5, 2.25]]
  }
}
```

Unit Test — Tabular

```
#! /usr/bin/env groovy

@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification
import spock.lang.Unroll

class functionsTest extends Specification {
  @Unroll def 'square always returns the square of the numeric parameter'() {
    expect:
    Functions.square(x) == r
    where:
    x | r
    0 | 0
    1 | 1
    2 | 4
    3 | 9
    1.5 | 2.25
  }
}
```

Exceptions

Code Under Test

```
class Exceptional {
  def trySomething() {
    throw new RuntimeException('Stuff happens.')
  }
}
```

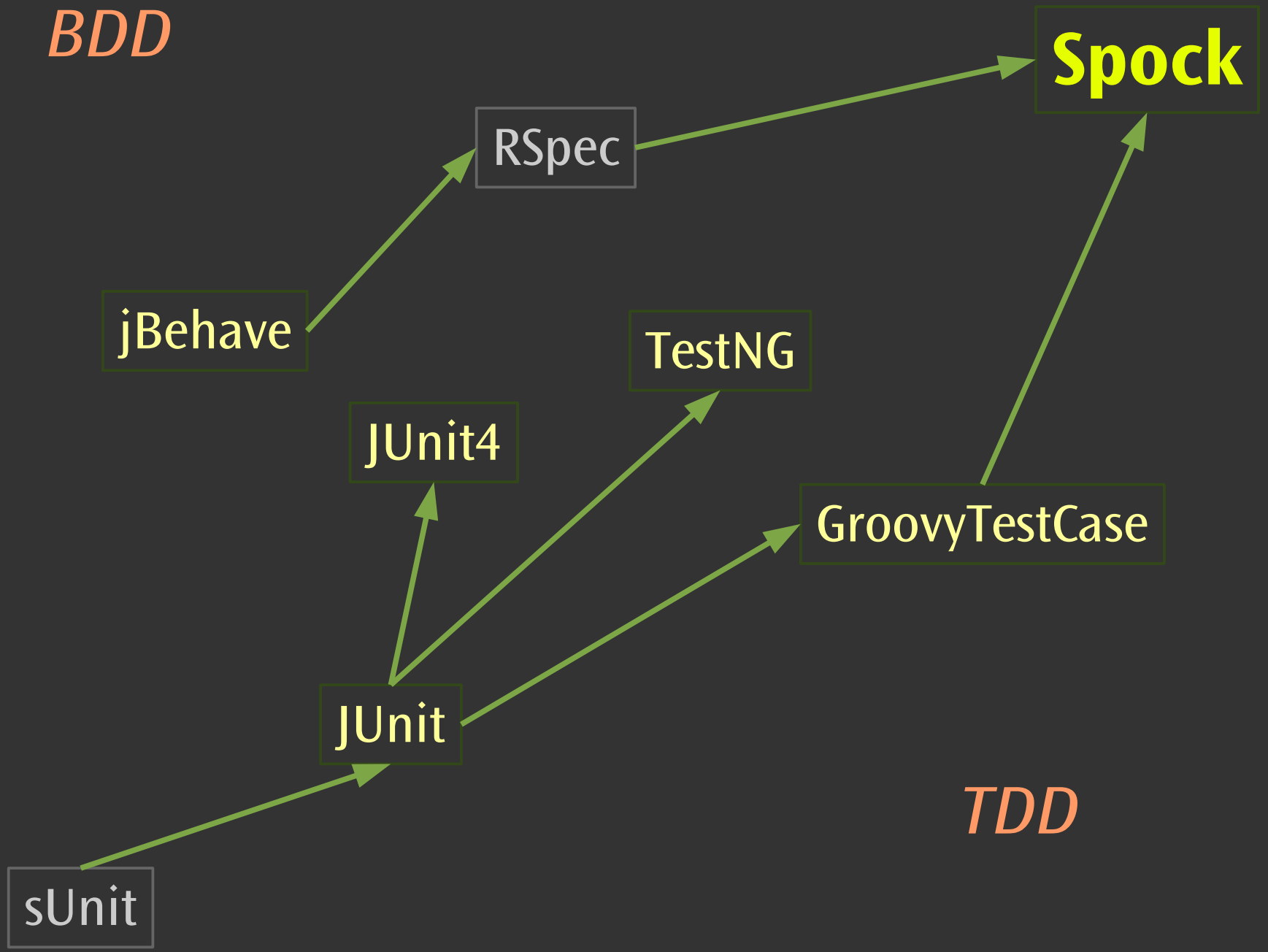
Exceptional.groovy

Unit Test

```
#! /usr/bin/env groovy

@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class ExceptionalTest extends Specification {
  def 'trying something always results in an exception'() {
    given:
     final e = new Exceptional ()
    when:
     e.trySomething()
    then:
     thrown(RuntimeException)
  }
}
```

Now we can do data validation
and
testing of error situations.

Being More Adventurous



Specify Behaviours - 1/4

```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class StackSpecification extends Specification {
  def 'newly created stacks are empty'() {
    given: 'a newly created stack'
    expect: 'the resulting stack to be empty.'
  }
```

Specify Behaviours - 2/4

```
def 'removing an item from a non-empty stack gives a value and changes the stack.'() {
  given: 'a new stack'
  and: 'an item to put on the stack'
  when: 'the item is added'
  then: 'the stack is not empty'
  when: 'an item is removed'
  then: 'the item we retrieved is the original and the stack is empty'
 }
}
```

Specify Behaviours - 3/4

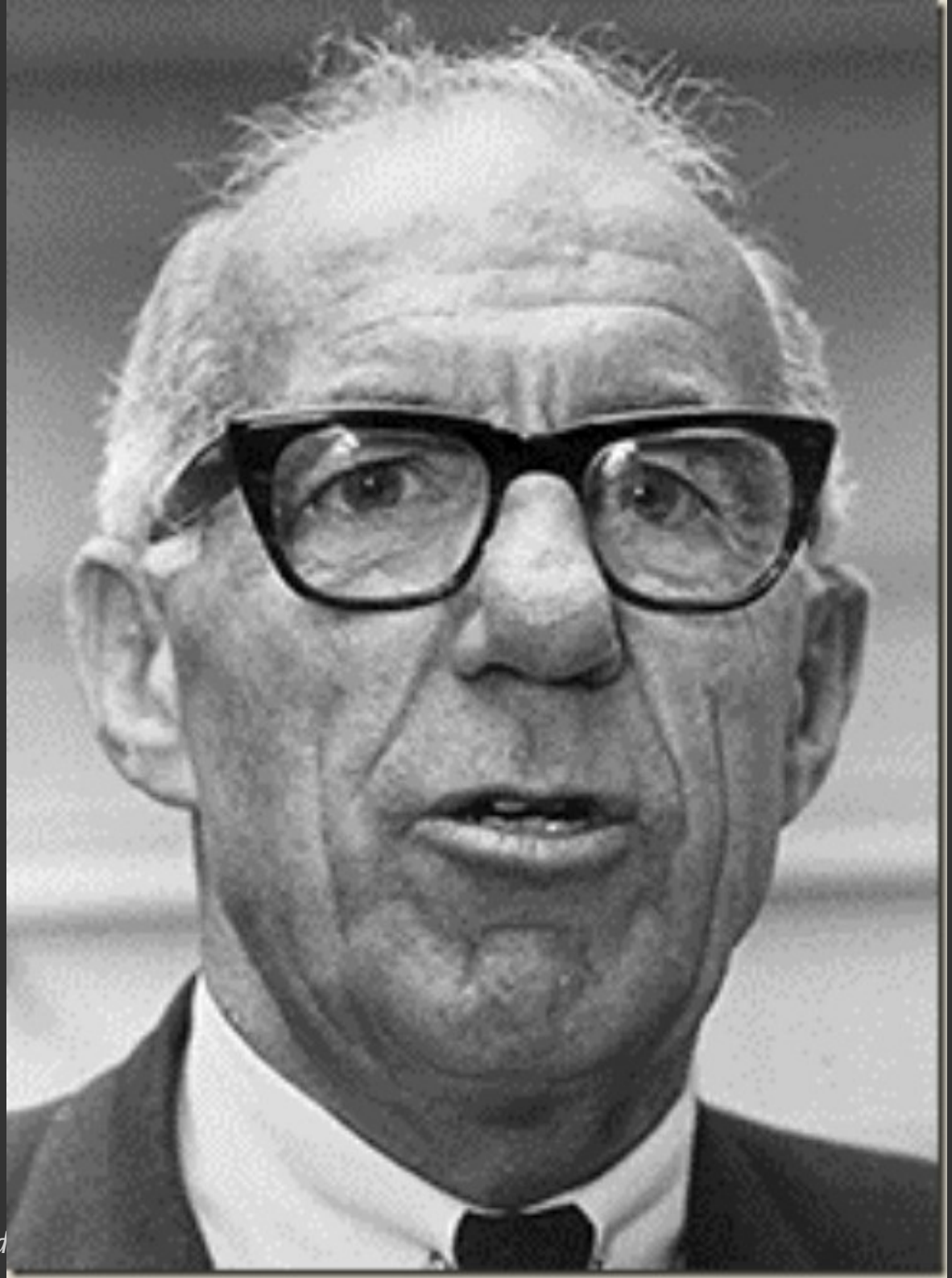
```
@Grab('org.spockframework:spock-core:1.0-groovy-2.4')
import spock.lang.Specification

class StackSpecification extends Specification {
  def 'newly created stacks are empty'() {
    given: 'a newly created stack'
    def stack = new Stack ()
    expect: 'the resulting stack to be empty.'
    stack.size() == 0
  }
```

Specify Behaviours - 4/4

```
def 'removing an item from a non-empty stack gives a value and changes the stack.'() {
  given: 'a new stack'
  def stack = new Stack ()
  and: 'an item to put on the stack'
  def item = 25
  and: 'a variable to store the result of activity'
  def result
  when: 'the item is added'
  stack.push(item)
  then: 'the stack is not empty'
  stack.size() == 1
  when: 'an item is removed'
  result = stack.pop()
  then: 'the item we retrieved is the original and the stack is empty'
  result == item && stack.size() == 0
 }
}
```

Closing

Hopefully everyone has had some fun
and
learnt some useful things.



The End

Spocktacular Testing

Russel Winder

email: russel@winder.org.uk
xmpp: russel@winder.org.uk
twitter: @russel_winder
Web: http://www.russel.org.uk