



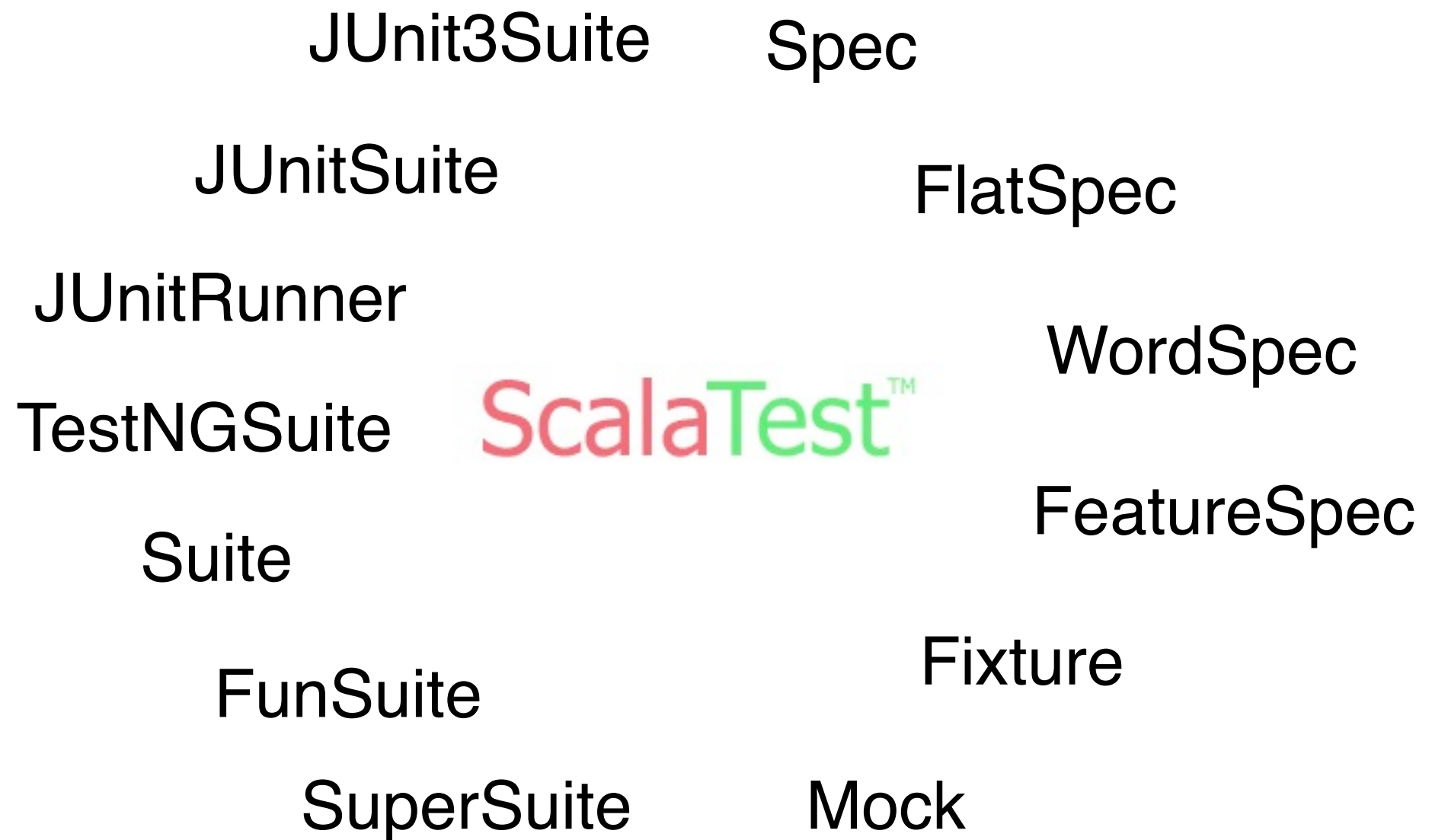
ScalaTest

Ole Christian Rynning

BEKK

<http://github.com/oc/fagdag1109>

ScalaTest



JUnit3 Style

```
class JUnitExampleSuite extends JUnit3Suite {  
  var service:SomeService = _  
  
  override def setUp { service = new SomeService() }  
  
  def testJUnitStyle() {  
    val result = service.invoke()  
    assertEquals("Some data!", data.computeData)  
    assertTrue(data.isValid)  
  }  
}
```

JUnit4 Style

```
class Junit4ExampleSuite extends JUnitSuite {  
  var service:SomeService = _  
  
  @Before def setUp { service = new SomeService() }  
  
  @Test def shouldDoSomethingJUnit4Style() {  
    val result = service.invoke()  
    assertEquals("Some data!", data.computeData)  
    assertTrue(data.isValid)  
  }  
}
```

TestNG Style

```
class TestNGExampleSuite extends TestNGSuite {  
  var service: SomeService = _  
  
  @Configuration { val beforeTestMethod = true }  
  def setUp { service = new SomeService() }  
  
  @Test { val groups = Array("no.bekk.groups.SlowAssTest") }  
  def shouldDoSomethingJUnit4Style() {  
    val result = service.invoke()  
    assertEquals("Some data!", data.computeData)  
    assertTrue(data.isValid)  
  }  
}
```

Suite

```
class SimpleSuite extends Suite {  
  def testAddition() {  
    val sum = 1 + 1  
    assert(sum === 2)  
    assert(sum + 2 === 4)  
  }  
  
  def testSubtraction() {  
    val diff = 4 - 1  
    assert(diff === 3)  
    assert(diff - 2 === 1)  
  }  
}
```

JUnitRunner (trunk)

```
import org.junit.runner.RunWith
import org.scalatest.junit.JUnitRunner
import org.scalatest.FunSuite

@RunWith(classOf[JUnitRunner])
class SimpleSuite extends Suite {
  def testAddition() {
    val sum = 1 + 1
    assert(sum == 2)
    assert(sum + 2 == 4)
  }

  def testSubtraction() {
    val diff = 4 - 1
    assert(diff == 3)
    assert(diff - 2 == 1)
  }
}
```


JUnit4Runner (github.com/teigen)

```
import org.junit.runner.RunWith
import com.jteigen.scalatest.JUnit4Runner
import org.scalatest.FunSuite
```

```
@RunWith(classOf[JUnit4Runner])
class SimpleSuite extends Suite {
  def testAddition() {
    val sum = 1 + 1
    assert(sum === 2)
    assert(sum + 2 === 4)
  }

  def testSubtraction() {
    val diff = 4 - 1
    assert(diff === 3)
    assert(diff - 2 === 1)
  }
}
```

Functional Suite

```
class SimpleFunSuite extends FunSuite {  
  test("addition") {  
    val sum = 1 + 1  
    assert(sum === 2)  
    assert(sum + 2 === 4)  
  }  
  
  test("subtraction") {  
    val diff = 4 - 1  
    assert(diff === 3)  
    assert(diff - 2 === 1)  
  }  
}
```

BDD Style(s)!

```
class PizzaSpec extends Spec with ShouldMatchers {  
  describe("Hungerfactor") {  
    it ("should order 4 pizzas for 10 people") {  
      Order(10).numberOfPizzas should be === 4  
    }  
    it("should order 4 pizzas for 9 people") {  
      Order(8).numberOfPizzas should be === 4  
    }  
    it("should order 3 pizzas for 8 people") {  
      Order(8).numberOfPizzas should be === 3  
    }  
  }  
}
```

Men først... Litt om ScalaTest's
syntaks og Matchers

Generelle Assertions

```
val left = 2
val right = 1
assert(left == right)
=> TestFailedException (fail)
```

```
val left = 2
val right = 1
assert(left === right)
=> TestFailedException("2 did not equal 1")
```

```
val a = 5
val b = 2
expect(2) { a - b }
=> TestFailedException("Expected 2, but got 3")
```

Generell Exception Testing (intercept)

```
try{
  service.validateParameter("Invalid!") // SomeException
  fail()
}
catch { case _: SomeException => //... }
```

```
val exception = intercept[SomeException] {
  service.validateParameter("Invalid!")
}
assert(exception.getMessage === "Some exception: Illegal parameter" )
```

```
it "should validate parameters" in {
  evaluating {
    service.validateParameter("Invalid!")
  } should produce [SomeException]
}
```

ShouldMatchers

```
val object = "abc"  
object should have length (3)
```

```
val emptySet = Set(1, 2, 3)  
emptySet should not be 'empty  
Set() should be (empty)
```

```
val str = "Hello world!"  
str should startWith regex "Hel*o"  
str should endWith regex "[wW]orld!?"  
str should include substring "world"  
str should fullyMatch regex ".*"  
str should equal ("Hello world!")
```

```
val n = 1  
n should be < 7  
n should be > 0  
n should be <= 7  
n should be >= 0
```

```
val o1 = new Object  
val o2 = new Object  
o1 should not be theSameInstanceAs (o2)
```

For de observante...

```
str should fullyMatch regex ".*"
```

```
str should equal ("Hello world!")
```


Best practice

~~str should fullyMatch regex ".*"~~

str should equal ("Hello world!")

Bruk det alltid!

```
str should fullyMatch regex (".*")
```

```
str should equal ("Hello world!")
```

MustMatchers

```
str must fullyMatch regex (".*")
```

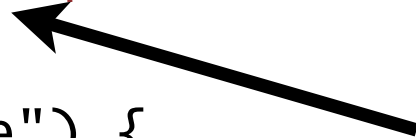
```
str must equal ("Hello world!")
```

```
trait ShouldMatchers extends Matchers with ShouldVerb
```

```
trait MustMatchers extends Matchers with MustVerb
```

Tilbake til BDD-eksempelet (RSpec style)

```
class PizzaSpec extends Spec with ShouldMatchers {  
  describe("Hungerfactor") {  
    it ("should order 4 pizzas for 10 people") {  
      Order(10).numberOfPizzas should be === 4  
    }  
    it("should order 4 pizzas for 9 people") {  
      Order(8).numberOfPizzas should be == 4  
    }  
    it("should order 3 pizzas for 8 people") {  
      Order(8).numberOfPizzas should equal (3)  
    }  
  }  
}
```



OBS!

FlatSpec (Enklere variant)

```
        extends Suite with ShouldVerb with MustVerb with CanVerb
class StackSpec extends FlatSpec {
  behavior of "A Stack"

  it should "pop values in last-in-first-out order" in {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    assert(stack.pop() === 2)
    assert(stack.pop() === 1)
  }

  it should "throw NoSuchElementException if an empty stack is popped" in {
    val emptyStack = new Stack[String]
    intercept[NoSuchElementException] {
      emptyStack.pop()
    }
  }
}
```

FeatureSpec (Cucumber light)

```
class PizzaFeatureSpec extends FeatureSpec with GivenWhenThen with ShouldVerb {
  feature("Order pizzas") {
    info("As an arranger")
    info("I want to optimize my pizza orders")
    info("So that I can satisfy attendees with minimal effort")

    scenario("order is called for 10 attendees") {
      given("I have a weighted menu with two pizzas")
      val menu = Orderer.Menu = Pizza("first", 150) * 10 **
                                Pizza("second", 150) * 1) toArray

      when("when I generate an order")
      val result = Orderer.order(10)

      then("I should order four pizzas")
      result.numberOfPizzas should be === (4)

      and("the price of the order should be less than 1200")
      result.price should be <= (1200)
    }
  }
}
```

Fixtures

```
class PizzaFixtureSuite extends FixtureFunSuite {  
  
  type FixtureParam = Menu[Pizza]  
  
  def withFixture(test: OneArgTest) {  
    val orderer = new Orderer(10)  
    val orderer.menu = Pizza("one", 149) ++ Pizza("two", 189)  
    test(orderer.menu)  
  }  
  
  test("fixture has has three Pizzas") { fixture =>  
    val menu = fixture  
    menu.append(Pizza("three", 169))  
    assert(menu.size == 3)  
  }  
  
  test("fixture has two Pizzas") { fixture =>  
    val menu = fixture  
    assert(menu.size == 2)  
  }  
  
}
```

Mocking m/Mockito (eksperimentelt!)

```
class ExampleSpec extends Spec with ShouldMatchers with MockitoSugar {  
  
    val mockService = mock[SomeService]  
  
    describe("Some Service invokation") {  
        it("should do something three times") {  
            mockService.invoke  
  
            // verify(mockService, times(3)).something(anyString)  
            mockService.something(anyString) should be invoked (3 times)  
        }  
    }  
}
```


Referanser

- <http://www.scalatest.org>
- <http://github.com/teigen/scalatest-junit4runner>
- <http://github.com/teigen/maven-scalatest-plugin>
- <http://www.artima.com/forums/flat.jsp?forum=106&thread=246279>