JUnit

ScalaTest

specs2

Testa med Scala

Cucumber

ScalaTest

```
class ReversiSuite extends AssertionsForJUnit {

  var board: Board = _

  @Before
  def setup() {
    board = new Board
    board.startGame
  }

  @Test
  def startBoardIsSetupCorrectly() {
    assertEquals(Black, board.squares(3)(3))
    assertEquals(White, board.squares(3)(4))
    assertEquals(White, board.squares(4)(3))
    assertEquals(Black, board.squares(4)(4))
  }
```

Olika Spec-stilar

Spec

```
class StackSpec extends Spec {
  describe("A Stack") {
    it("should pop values in last-in-first-out order") {
```

FlatSpec

```
class StackSpec extends FlatSpec {
  behavior of "A Stack"
  it should "pop values in last-in-first-out order" in {
```

WordSpec

```
class StackSpec extends WordSpec {
  "A Stack" should {
    "pop values in last-in-first-out order" in {
```

FeatureSpec

```
class StackFeatureSpec extends FeatureSpec with GivenWhenThen {
  feature("The user can pop an element off the top of the stack")
{
    info("As a programmer")
    info("I want to be able to pop items off the stack")
    info("So that I can get them in last-in-first-out order")

    scenario("pop is invoked on a non-empty stack") {

      given("a non-empty stack")
      val stack = new Stack[Int]
      stack.push(1)
      stack.push(2)
      val oldSize = stack.size

      when("when pop is invoked on the stack")
      val result = stack.pop()

      then("the most recently pushed element should be returned")
      assert(result === 2)
    }
```

specs²

```
class MontySpec extends Specification {

  val game = new Game(List(Goat(), Goat(), Car()))
  val player = new Player(game)

  "The Game" should {

    "make Monty choose another door than the player" in {
      player.chooseDoor(0)
      game.montyDoor must_!= game.playerDoor
    }

    "know that the player won if he chooses the Car" in {
      player.chooseDoor(2)
      game.won must_== true
    }
```

```
"A simulation" should {
  "show that switching will win two times out of three" in {
    val thousandGames = for(i <- 1 to 1000) yield {
      val game = new Game
      val player = new Player(game)
      player.chooseDoor(0)
      player.switch()
      game.won()
    }
    thousandGames.count(_ == true) must be closeTo(667 +/- 67)
  }
}
```

- Matchers for Any
- Option / Either matchers
- String matchers
- Numeric matchers
- Exception matchers
- Iterable matchers
- Map matchers
- Xml matchers
- Json matchers
- File matchers
- Scalaz matchers
- Result matchers
- Interpreter matchers

```
import org.specs2.mock._
  class MockitoSpec extends Specification { def is =

     "A java list can be mocked"                                    ^
       "You can make it return a stubbed value"                     ! c().stub^
     "You can verify that a method was called"                      ! c().verify^
     "You can verify that a method was not called"                  ! c().verify2^
                                                                    end
```

```scala
case class c() extends Mockito {
  // a concrete class would be mocked with: mock(new java.util.LinkedList[String])
  val m = mock[java.util.List[String]]
  def stub = {
    m.get(0) returns "one"                // stub a method call with a return value
    m.get(0) must_== "one"                // call the method
  }
  def verify = {
    m.get(0) returns "one"                // stub a method call with a return value
    m.get(0)                              // call the method
    there was one(m).get(0)               // verify that the call happened
  }
  def verify2 = there was no(m).get(0) // verify that the call never happened
}
```

ScalaCheck

```
object StringSpecification extends Properties("String") {
  property("startsWith") = Prop.forAll((a: String, b: String) =>
    (a+b).startsWith(a))

  property("endsWith") = Prop.forAll((a: String, b: String) =>
    (a+b).endsWith(b))

  // Is this really always true?
  property("concat") = Prop.forAll((a: String, b: String) =>
    (a+b).length > a.length && (a+b).length > b.length)
```

```
scala -cp .:scalacheck-1.8.jar StringSpecification
+ String.startsWith: OK, passed 100 tests.
+ String.endsWith: OK, passed 100 tests.
! String.concat: Falsified after 0 passed tests.
> ARG_0:
> ARG_1:
```

```
Gen.choose(10,20)
```

```
Gen.oneOf('A', 'E', 'I', 'O', 'U', 'Y')
```

```
val vowel = Gen.frequency(
  (3, 'A'),
  (4, 'E'),
  (2, 'I'),
  (3, 'O'),
  (1, 'U'),
  (1, 'Y')
)
```

```
Gen.containerOf[List,Int](Gen.oneOf(1, 3, 5))
```

```
Arbitrary.arbitrary[Int] suchThat (_ % 2 == 0)
```

```
val smallOdds = Gen.choose(1,100) suchThat (_ % 2 == 1)

Prop.forAll(smallOdds,smallOdds)((a:Int,b:Int) =>
  { a+b == b+a }).check
```


Cucumber

```
Feature: Monty Hall Game
  In order to win a Car
  As a Player
  I want to be able to play the Monty Hall Game

  Background:
      Given there exists a Game
        And there exists a Player

Scenario: Monty should display a Goat
     When the Player picks a Door
     Then Monty should display a Goat
```

```
Feature: Monty Hall Game
  In order to win a Car
  As a Player
  I want to be able to play the Monty Hall Game
```

```
import cuke4duke.{EN, ScalaDsl}
import org.scalatest.matchers.ShouldMatchers

class MontySteps extends ScalaDsl with EN with
ShouldMatchers {
```

```
Background:
    Given there exists a Game
      And there exists a Player
```

```
var game:Game = _
var player:Player = _

Given("^there exists a Game$") {
  game = new Game
}

And("^there exists a Player$") {
  player = new Player(game)
}
```

```
Scenario: Monty should display a Goat
    When the Player picks a Door
    Then Monty should display a Goat
```

```
When("""^the Player picks Door no (\d+)$""") { doorNo:Int =>
  player.chooseDoor(doorNo - 1)
}

When("^the Player picks a Door$") {
  When("the Player picks Door no " + (Random.nextInt(2) + 1))
  game.playerDoor should not be -1
}

Then("""^Monty should display a (\w+)$""") { door:String =>
  game.doors(game.montyDoor) should be(stringToDoor(door))
}
```

> *A step description should never contain regexen, CSS or XPath selectors, any kind of code or data structure. It should be easily understood just by reading the description.*

Enhetstester för lågnivå
Cucumber för acceptanstest

project/plugins/Plugin.scala

```
import sbt._

class Plugins(info: ProjectInfo) extends PluginDefinition(info) {

  val templemoreRepo = "templemore repo" at "http://templemore.co.uk/repo"
  val cucumberPlugin = "templemore" % "cucumber-sbt-plugin" % "0.4.1"
}
```

project/build/TheProj.scala

```
import sbt._
import templemore.sbt.CucumberProject

class TheProj(info: ProjectInfo) extends DefaultProject(info) with
CucumberProject {

  val scalaToolsSnapshots = ScalaToolsSnapshots

  val scalatest = "org.scalatest" % "scalatest" %  "1.3" % "test"
  val specs = "org.scala-tools.testing" %% "specs" %  "1.6.6" % "test"
}
```

http://www.scalatest.org/
http://etorreborre.github.com/specs2/
http://code.google.com/p/scalacheck/

http://cukes.info/
https://github.com/aslakhellesoy/cuke4duke/wiki
http://elabs.se/blog/15-you-re-cuking-it-wrong

http://skipoleschris.blogspot.com/2010/11/using-cucumber-with-scala-and-sbt.html
https://github.com/skipoleschris/cucumber-sbt-plugin