Jfokus 2019

mindbox

Spock vs JUnit 5 - Clash of the Titans

Marcin Zajączkowski

Stockholm, 6 February 2019

@SolidSoftBlog

About me

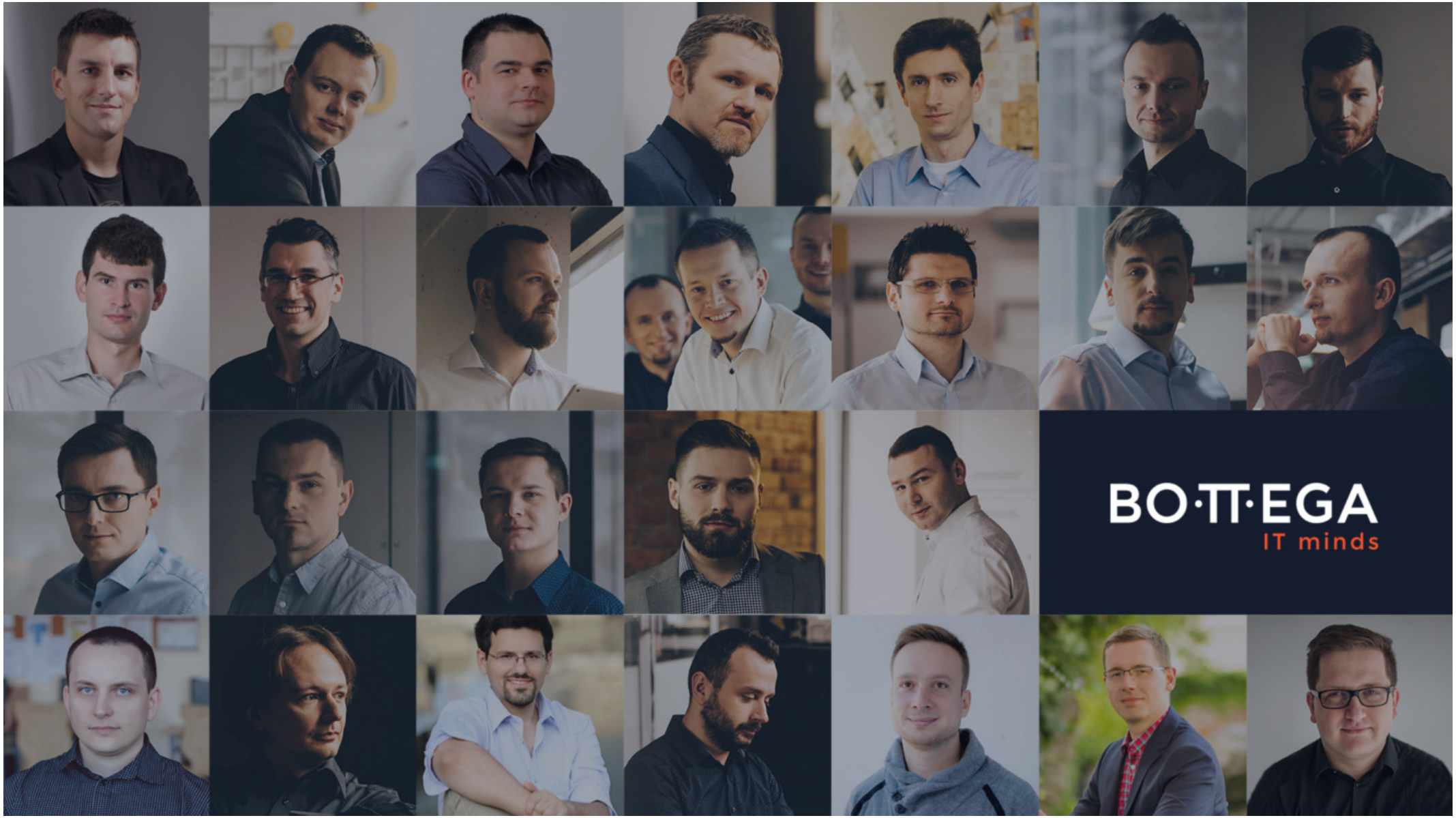
- Areas of expertise
  - **Automatic Testing / TDD**
  - Software Craftsmanship / Code Quality
  - Deployment Automation / Continuous Delivery
  - Concurrency / Parallel Computing / Reactive Systems

- FOSS projects author and contributor
- blogger
- trainer

Comprehensive Kubernetes & Openshift services

with Continuous Delivery with Jenkins

https://mindboxgroup.com/
cloud@mindboxgroup.com


BO·TT·EGA
IT minds

Presentation goal

Help you decide what is the best testing framework for Java code

Presentation plan

mindbox

- historical outline
- selected JUnit 5 & Spock features comparison
- summary

Version statement

This comparison is valid for JUnit 5.4.0-RC2 and Spock 1.3-RC1

Historical outline

Unit testing in Java - historical outline

mindbox

- JUnit - first xUnit for Java - 2000

Unit testing in Java - historical outline

mindbox

- JUnit - first xUnit for Java - 2000
- TestNG - Java 5 leveraged in tests - 2004
  - with some unique (at the time) features

Unit testing in Java - historical outline

mindbox

- JUnit - first xUnit for Java - 2000
- TestNG - Java 5 leveraged in tests - 2004
  - with some unique (at the time) features
- JUnit 4 - Java 5 support - 2006
  - catching up TestNG features over years
  - de facto standard, steady evolution rather than revolution

Unit testing in Java - historical outline

mindbox

- JUnit - first xUnit for Java - 2000
- TestNG - Java 5 leveraged in tests - 2004
  - with some unique (at the time) features
- JUnit 4 - Java 5 support - 2006
  - catching up TestNG features over years
  - de facto standard, steady evolution rather than revolution
- Spock - revamped test creation - 2009
  - power of Groovy under the hood

Unit testing in Java - historical outline

mindbox

- JUnit - first xUnit for Java - 2000
- TestNG - Java 5 leveraged in tests - 2004
  - with some unique (at the time) features
- JUnit 4 - Java 5 support - 2006
  - catching up TestNG features over years
  - de facto standard, steady evolution rather than revolution
- Spock - revamped test creation - 2009
  - power of Groovy under the hood
- JUnit 5 - redesigned and rewritten from scratch - 2017
  - new king of the hill?

mindbox

Development

Development

mindbox

inception year:

number of GitHub stars:

number of commits:

development activity:

number of **active** committers:

number of contributors (ever):

tool support:

Development

mindbox

	JUnit 5 (v5.3)
inception year:	2015
number of GitHub stars:	~2,5K
number of commits:	~4,9K
development activity:	high (young project)
number of **active** committers:	3
number of contributors (ever):	~90
tool support:	very good (trending up)

Development

mindbox

	JUnit 5 (v5.3)	**Spock** (v1.2)
inception year:	2015	2009
number of GitHub stars:	~2,5K	~2K
number of commits:	~4,9K	~2,4K
development activity:	high (young project)	medium (mature project)
number of **active** committers:	3	2
number of contributors (ever):	~90	~70
tool support:	very good (trending up)	good

mindbox

Development

partial verdict

mindbox

Development

partial verdict

Tool support

Tool support

mindbox

Java 11:

IntelliJ IDEA:

Eclipse:

Netbeans:

Maven:

Gradle:

SonarQube:

PIT - mutation testing:

Tool support

mindbox

	JUnit 5	Spock
Java 11:	*very good*	*good* *(with Groovy 2.5.3+)*
IntelliJ IDEA:	built-in	built-in (some Groovy limitations)
Eclipse:	built-in	built-in (some Groovy limitations)
Netbeans:	10.0+ (Maven only)	unknown
Maven:	plugin (Surefire)	plugin (GMavenPlus for Groovy)
Gradle:	built-in	built-in
SonarQube:	built-in	plugin (official for Groovy)
PIT - mutation testing:	plugin (official)	built-in

Tool support

partial verdict

mindbox

Tool support

partial verdict

Test structure

Test structure - BDD-like specification - in general

mindbox

- clear separation in 3 blocks with predefined responsibility
  - given - creation, initialization and stubbing
  - when - operation to test
  - then - assertion and interaction verification

Test structure - BDD-like specification - in general

mindbox

- clear separation in 3 blocks with predefined responsibility
  - given - creation, initialization and stubbing
  - when - operation to test
  - then - assertion and interaction verification
- unified (procedures for) test creation
- improved readability
- easier to get started with writing (good) test
  - especially for less experienced people

Test structure - JUnit 5

mindbox

```
class SimpleCalculatorTest {

    @Test
    void shouldAddTwoNumbers() {
        //given
        Calculator calculator = new Calculator();
        //when
        int result = calculator.add(1, 2);
        //then
        assertEquals(3, result);
    }
}
```

- just plain code comments
- easy to forgot if template is not used
- can be easily lost/misplaced in refactoring

Test structure - Spock

```
class SimpleCalculatorSpec extends Specification {

    def "should add two numbers"() {
        given:
            Calculator calculator = new Calculator()
        when:
            int result = calculator.add(1, 2)
        then:
            result == 3
    }
}
```

- `[given]/when/then` (or `expect`) required to compile code
- especially handy with predefined test template

Test structure

partial verdict

mindbox

Test structure

partial verdict

mindbox

Exception testing

Exception testing - in general

mindbox

- verification if proper exception was thrown
- often in particular line in test
- usually accompanied by verification of
  - proper error message
  - extra field(s) set in exception instance

Exception testing - JUnit 5

- ~~`@Test(expected = NullPointerException.class)` for one-liners~~

Exception testing - JUnit 5

mindbox

- ~~`@Test(expected = NullPointerException.class)` for one-liners~~
- `assertThrows()` for fine grained assertions
  - similar to `catchThrowable()` from AssertJ

```
@Test
void shouldThrowBusinessExceptionOnCommunicationProblem() {
    //when
    Executable e = () -> client.sendPing(TEST_REQUEST_ID)
    //then
    CommunicationException thrown = assertThrows(CommunicationException.class, e);
    assertEquals("Communication problem when sending request with id: " + TEST_REQUEST_ID,
                 thrown.getMessage());
    assertEquals(TEST_REQUEST_ID, thrown.getRequestId());
}
```

- compact syntax (thanks to lambda expression)
- further assertions possible on returned instance

Exception testing - Spock

- `@FailsWith(NullPointerException)` for one-liners

Exception testing - Spock

- `@FailsWith(NullPointerException)` for one-liners
- `thrown()` method to catch exceptions in `when` block

```
def "should capture exception"() {
    when:
        client.sendPing(TEST_REQUEST_ID)
    then:
        CommunicationException e = thrown()
        e.message == "Communication problem when sending request with id: $TEST_REQUEST_ID"
        e.requestId == TEST_REQUEST_ID
}
```

- further assertions possible on returned instance
- very compact syntax (thanks to Groovy AST transformations)
- smart type inference
- `Exceptions` utility class to deal with cause chain

mindbox

Exception testing

partial verdict

mindbox

Exception testing

partial verdict

mindbox

Conditional test execution

Conditional test execution - in general

mindbox

- test executed only if given condition is (not) met

Conditional test execution - in general

mindbox

- test executed only if given condition is (not) met

- common cases
  - particular Java version
    - reflection hack to use newer version features
    - compatibility testing for lower version
  - specific operating system
    - notifications about changed files on Mac are much delayed
    - testing symlinks on Windows makes no sense
  - tests executed only on CI server, stage environment, ...

JfokuS 2019

Conditional test execution - JUnit

mindbox

- annotation-based `@Enabled*/@Disabled*` conditions
- predefined set of conditions
  - JVM version, operating system
  - system property, environment variable

```
@Test
@DisabledOnOs(OS.WINDOWS)
void shouldTestSymlinksBasedLogic() {
    ...
}

@Test
@EnabledIfSystemProperty(named = "os.arch", matches = ".*32.*")
void shouldBeRunOn32BitSystems() {
    ...
}
```

Conditional test execution - JUnit - cont'd

mindbox

- good code completion (enums)
- easily composable as meta-annotations

Conditional test execution - JUnit - cont'd

mindbox

- good code completion (enums)
- easily composable as meta-annotations
- experimental support for custom logic in script-based conditions

```
@Test
@DisabledIf("'Travis' == systemEnvironment.get('CI_SERVER')")
void shouldBeDisabledOnTravis() {
    ...
}
```

Conditional test execution - Spock

mindbox

- `@Requires`/`@IgnoreIf` built-in extensions
- predefined set of conditions
  - JVM version, operating system
  - system property, environment variable

```
@IgnoreIf({ !jvm.java8Compatible })
def "should return empty Optional by default for unstubbed methods with Java 8+"() { ... }
```

Conditional test execution - Spock

mindbox

- `@Requires`/`@IgnoreIf` built-in extensions
- predefined set of conditions
  - JVM version, operating system
  - system property, environment variable

```
@IgnoreIf({ !jvm.java8Compatible })
def "should return empty Optional by default for unstubbed methods with Java 8+"() { ... }
```

- no code completion by default (possible with small trick)
- custom logic in Groovy closure

```
@Requires({ isStrongCryptographyEnabled() })    //custom static method
def "should test strong cryptography-based features"() { ... }
```

mindbox

Conditional test execution

partial verdict

mindbox

Conditional test execution

partial verdict

mindbox

Mocking

Mocking - in general

mindbox

- testing with mocks (stubs) instead real collaborators
- stubbing call executions
- verifying interactions

Mocking - JUnit 5

mindbox

- no built-in mocking framework
  - Mockito - first port of call

```
@Test
public void should_not_call_remote_service_if_found_in_cache() {
    //given
    given(cacheMock.getCachedOperator(CACHED_MOBILE_NUMBER)).willReturn(Optional.of(PLUS));
    //when
    service.checkOperator(CACHED_MOBILE_NUMBER);
    //then
    verify(wsMock, never()).checkOperator(CACHED_MOBILE_NUMBER);
}
```

Mocking - JUnit 5

mindbox

- no built-in mocking framework
  - Mockito - first port of call

```
@Test
public void should_not_call_remote_service_if_found_in_cache() {
    //given
    given(cacheMock.getCachedOperator(CACHED_MOBILE_NUMBER)).willReturn(Optional.of(PLUS));
    //when
    service.checkOperator(CACHED_MOBILE_NUMBER);
    //then
    verify(wsMock, never()).checkOperator(CACHED_MOBILE_NUMBER);
}
```

- industrial standard
- rich set of features
- first-class support for integration testing with Spring Boot (`@MockBean`)

JFokus 2019

Mocking - Spock

mindbox

- built-in mocking subsystem

```
def "should not hit remote service if found in cache"() {
    given:
        cacheMock.getCachedOperator(CACHED_MOBILE_NUMBER) >> Optional.of(PLUS)
    when:
        service.checkOperator(CACHED_MOBILE_NUMBER)
    then:
        0 * wsMock.checkOperator(CACHED_MOBILE_NUMBER)
}
```

Mocking - Spock

- built-in mocking subsystem

```
def "should not hit remote service if found in cache"() {
    given:
        cacheMock.getCachedOperator(CACHED_MOBILE_NUMBER) >> Optional.of(PLUS)
    when:
        service.checkOperator(CACHED_MOBILE_NUMBER)
    then:
        0 * wsMock.checkOperator(CACHED_MOBILE_NUMBER)
}
```

- extra short and more meaningful syntax
  - thanks to Groovy operator overloading & AST transformations
  - unbeatable by anything written in pure Java *
- Spring Boot support starting with Spock 1.2
- its own limitations and quirks
  - Mockito can be used selectively if preferred/needed

mindbox

Mocking

partial verdict

mindbox

Mocking

partial verdict

Draw

mindbox

Parameterized tests

Parameterized tests - in general

mindbox

- one test for various input data
- reduce code duplication
  - can hide specific business use cases

Parameterized tests - JUnit 5

mindbox

- in JUnit 4 very late added and poorly designed

```
@ParameterizedTest
@CsvSource({"1, 2, 3", "-2, 3, 1", "-1, -2, -3"})
void shouldSumTwoIntegers(int x, int y, int expectedResult) {
    //when
    int result = calculator.add(x, y);
    //expect
    assertEquals(expectedResult, result);
}
```

- nice implicit argument conversion from String
  - for various types (date, file/path, currency, UUID, etc.)

Parameterized tests - JUnit 5

mindbox

- input parameters rendered in report with `@ParameterizedTest`
- complex declaration of custom data provider (method source)

```
@ParameterizedTest(name = "summing {0} and {1} should give {2}")
@MethodSource("integersProvider")
void shouldSumTwoIntegers(int x, int y, int expectedResult) {
    //when
    int result = calculator.add(x, y);
    //expect
    assertEquals(expectedResult, result);
}

private static Stream<Arguments> integersProvider() {
    return Stream.of(
            Arguments.of(1, 2, 3),
            Arguments.of(-2, 3, 1),
            Arguments.of(-1, -2, -3)
    );
}
```

Parameterized tests - Spock

mindbox

```
@Unroll
def "should sum two integers (#x + #y = #expectedResult)"() {
    when:
        int result = calculator.add(x, y)
    then:
        result == expectedResult
    where:
         x |  y || expectedResult
         1 |  2 ||  3
        -2 |  3 ||  1
        -1 | -2 || -3
}
```

- state of the art

Parameterized tests - Spock

mindbox

```
@Unroll
def "should sum two integers (#x + #y = #expectedResult)"() {
    when:
        int result = calculator.add(x, y)
    then:
        result == expectedResult
    where:
         x |  y || expectedResult
         1 |  2 ||  3
        -2 |  3 ||  1
        -1 | -2 || -3
}
```

- state of the art
- `where` keyword with table-like formatting
  - very readable and natural to read
- variables added implicitly (with proper type inferred by IDE)
- input parameters possible to use in test name directly (with `#var` syntax)

Jfokus 2019

Parameterized tests - Spock

mindbox

- data pipes and data providers for advanced use cases

```
@Unroll("#pesel is valid (#dbId)")
def "should validate PESEL correctness (CSV)"() {
    expect:
        sut.validate(pesel)
    where:
        [dbId, _, _, pesel] << readValidPeopleFromCSVFile()
                                    .readLines().collect { it.split(',') }
                                //ugly way to read CSV - don't do this
}
```

mindbox

Parameterized tests

partial verdict

mindbox

Parameterized tests

partial verdict

mindbox

Migration

Migration to JUnit 5 (from JUnit 4)

mindbox

- can coexist with JUnit 4 tests
  - dedicated submodule - `junit5-vintage`
- similar test structure
  - with new keywords, annotations, assertions, features
- new base class package

Migration to Spock (from JUnit 4)

mindbox

- can coexist with JUnit 4 tests
  - in fact Spock is a (sophisticated) JUnit runner
- unofficial tool for automatic test migration
- completely new test structure

mindbox

Migration

partial verdict

mindbox

Migration

partial verdict

Summary

Summary

- JUnit 5 - great progress over JUnit 4

Summary

mindbox

- JUnit 5 - great progress over JUnit 4
- many features (previously) unique to Spock now available in JUnit 5

Summary

mindbox

- JUnit 5 - great progress over JUnit 4
- many features (previously) unique to Spock now available in JUnit 5
- Spock still excels in some areas

Summary

mindbox

- JUnit 5 - great progress over JUnit 4
- many features (previously) unique to Spock now available in JUnit 5
- Spock still excels in some areas
- best choice is no longer obvious

Summary

mindbox

- JUnit 5 - great progress over JUnit 4
- many features (previously) unique to Spock now available in JUnit 5
- Spock still excels in some areas
- best choice is no longer obvious
  - it depends on individual preferences

Summary - features comparison

mindbox

	JUnit 5	Spock
development:	very good	good (new developers recently)
learning curve:	very good (similar to 4)	good (Groovy to grasp)
tool support:	very good (trending up)	good (weaker compile time checks)
test structure:	good	very good (BDD by default)
exception testing:	good	very good
conditional execution:	good (limited custom logic)	very good (custom logic)
mocking:	good (Mockito)	good (very compact, some quirks)
parameterized tests:	good	**very good** (exceptional!)
migration from JUnit 4:	very good	good

JFokus 2019

Summary - the choice is yours

mindbox

- if you prefer
  - old good Java as the only language
  - stability and being mainstream
  - stronger compile time error checking - **choose JUnit 5**

Summary - the choice is yours

mindbox

- if you prefer
  - old good Java as the only language
  - stability and being mainstream
  - stronger compile time error checking - **choose JUnit 5**

- if you favor
  - simplicity and readability
  - power of Groovy under the hood
  - beautiful parameterized tests and exception testing - **choose Spock**

Summary - the choice is yours

mindbox

- if you prefer
  - old good Java as the only language
  - stability and being mainstream
  - stronger compile time error checking - **choose JUnit 5**

- if you favor
  - simplicity and readability
  - power of Groovy under the hood
  - beautiful parameterized tests and exception testing - **choose Spock**

- **would be great to have only that kind of dilemma :-)**

Thank you!

(and remember about the feedback)

Marcin Zajączkowski

https://blog.solidsoft.info/
@SolidSoftBlog
m.zajaczkowski@gmail.com

OpenPGP: 0x48A84C5100F47FB6
06FA 6793 8DD1 7603 B007 5522 48A8 4C51 00F4 7FB6

Questions?

(and remember about the feedback)

Marcin Zajączkowski

https://blog.solidsoft.info/
@SolidSoftBlog
m.zajaczkowski@gmail.com

OpenPGP: 0x48A84C5100F47FB6
06FA 6793 8DD1 7603 B007 5522 48A8 4C51 00F4 7FB6