Specifications with Spock

By Bytecraft_

Agenda

- Introduction
- **Theory and examples**
  - Problems of xUnit testing
  - Basics of Spock
    - xUnit vs Spock
  - Idiomatic Spock & Groovy
    - Documenting with tests (specifications)
    - Eliminating duplication
    - Isolation
    - How to build test data
- **Hands-on exercises**

Who?

- Bytecraft_
    - Software Craft
        - https://manifesto.softwarecraftsmanship.org/
    - https://www.bytecraft.fi/
- Antti Ahonen

Unit testing

- Tests individual unit or collection of these units working as one [1, 2]

- A good unit test is [3]
  - maintainable
  - readable
  - isolated
  - single concern
  - minimal amount of repetition

But let's check a bad one first:
`fi.bytecraft.spock.algorithm.BadCurlyBracesCheckerTest`
And a better one:
`fi.bytecraft.spock.algorithm.CurlyBracesCheckerTest`

Problems with xUnit

Starting point

- Majority of developers find unit tests helpful in producing higher quality code [13]
- Majority of developers find unit tests helpful in understanding other people's code [13]

Problems?

- Developers are mainly trying to find realistic scenarios on what to test [11]
- Developers finding isolating of unit under test hard [11]
- Only half of the survey respondents enjoy writing unit tests [11, 12]
- Maintaining unit tests was found hard [11, 12]
- For 60.4% of developers, understanding unit tests is at least moderately difficult [14]
- Developers find updated documentation and comments in mtest cases useful, but writing comments to unit tests is rarely or never done [14]

Spock 101: Overview

Like xUnit, but enhanced for readability and maintainability

Behavior Driven Specification framework

- Describe the desired behavior of system under test
- Produces Spec files with feature methods
    - constructed with **Gherkin** blocks
        - **Given, When, Then**
- Dynamic features from Groovy-language
    - Data-Driven Testing
    - Mocks & stubs
    - Debug prints
    - Dynamic builders etc..

Spock 101: Building blocks

- **Gherkin**-blocks
- Separating different parts of test (feature method)
    - **Given**: For test context creating
    - **When**: For tested action
    - **Then**: For assertions against action results
- and a couple of extra blocks
    - **And**: Can be applied after any block to continue using the previous block
    - **Expect**:
        - For doing assertions on the initial context before action
        - or combining when and then in into one in concise action + assertion
    - **Where**: Data-Driven testing
- Check primer for more: http://spockframework.org/spock/docs/1.3/spock_primer.html

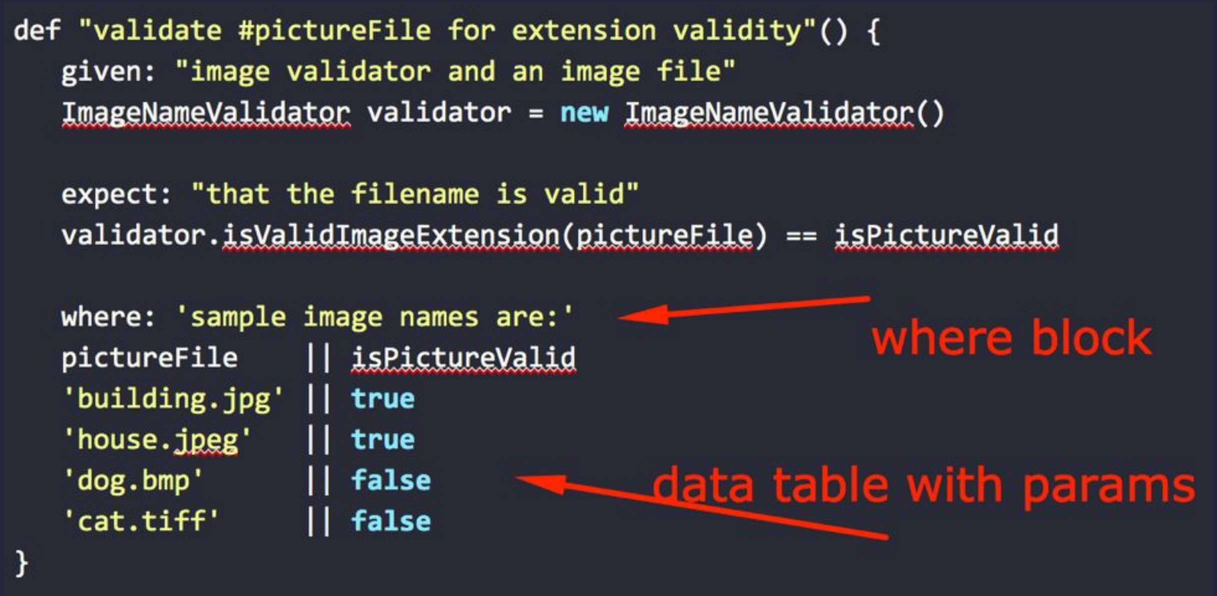
Spock 101: Simple example with Spock

- Gherkin in action
- Exception handling

```
Example specification can be found here
fi.bytecraft.spock.algorithm.Start_CurlyBracesCheckerSpec
```

Spock 101: Data Driven Testing

- Tabular format readable domain specific language (DSL)
- Remove repetition from code
- Test different parameter variations easily



```
def "validate #pictureFile for extension validity"() {
    given: "image validator and an image file"
    ImageNameValidator validator = new ImageNameValidator()

    expect: "that the filename is valid"
    validator.isValidImageExtension(pictureFile) == isPictureValid

    where: 'sample image names are:'
    pictureFile     || isPictureValid
    'building.jpg'  || true
    'house.jpeg'    || true
    'dog.bmp'       || false
    'cat.tiff'      || false
}
```

Spock 101: Data Driven Testing

- Let’s check the previous example in data driven style

  `Example specification can be found here`
  `fi.bytecraft.spock.algorithm.CurlyBracesCheckerSpec`

- And then it’s time to do some basic Spock unit testing **(SPECIFICATIONS + DATA DRIVEN TESTING)**

  `Example code to test can be found`

- `fi.bytecraft.spock.animals.Dog`
  - `Try out the basic building blocks with feature methods describing dog behaviour`
- `fi.bytecraft.spock.animals.Cat`
  - `Can you data drive the cat behaviour?`
  - `After this example, can you data drive the Dog behaviour?`

Spock 101: Simple integration testing

- Gherkin in action
- Simple test data creating
- Exception handling

```
Example specification can be found here
fi.bytecraft.spock.reviewservice.AddCommentISpec
```

Spock 101: Unit test Isolation

Same example service method, but now with unit test approach

- Mocking
- Stubbing
- Verifying mock object interactions
- Argument capturing
- More from here:
  - http://spockframework.org/spock/docs/1.3/interaction_based_testing.html

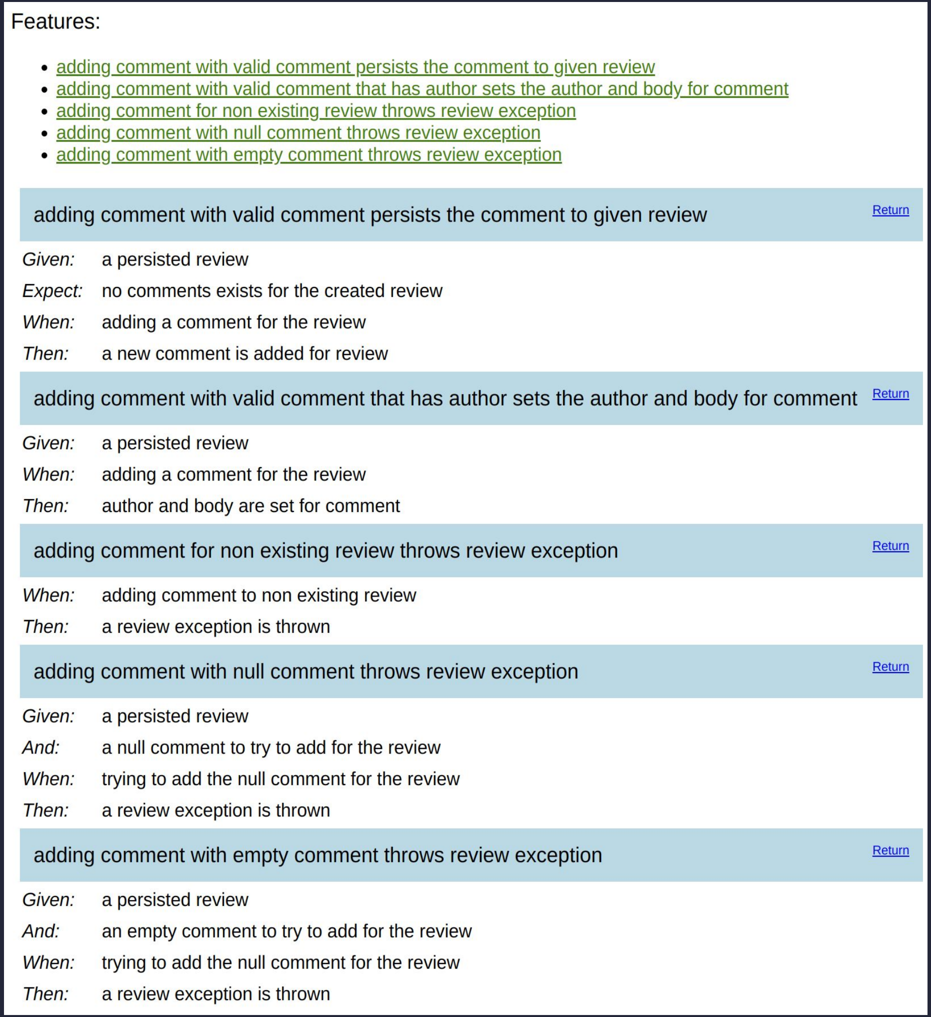
- Let’s check Spock isolation in practice

```
Example specification can be found here
fi.bytecraft.spock.reviewservice.AddCommentSpec
```

Spock 101: Documentation

Structured comments that generate **living documentation**

`Specification source for reports can be found here fi.bytecraft.spock.reviewservice.AddCommentISpec`



Features:

- adding comment with valid comment persists the comment to given review
- adding comment with valid comment that has author sets the author and body for comment
- adding comment for non existing review throws review exception
- adding comment with null comment throws review exception
- adding comment with empty comment throws review exception

adding comment with valid comment persists the comment to given review Return

Given:	a persisted review
Expect:	no comments exists for the created review
When:	adding a comment for the review
Then:	a new comment is added for review

adding comment with valid comment that has author sets the author and body for comment Return

Given:	a persisted review
When:	adding a comment for the review
Then:	author and body are set for comment

adding comment for non existing review throws review exception Return

When:	adding comment to non existing review
Then:	a review exception is thrown

adding comment with null comment throws review exception Return

Given:	a persisted review
And:	a null comment to try to add for the review
When:	trying to add the null comment for the review
Then:	a review exception is thrown

adding comment with empty comment throws review exception Return

Given:	a persisted review
And:	an empty comment to try to add for the review
When:	trying to add the null comment for the review
Then:	a review exception is thrown

How to build test data?

- Fixtures
- **Named constructor parameters**
- **Factory-pattern**
- **Builder-pattern**
- Use the actual production code operations

- **Build the minimal test data for test method under run**
- **Avoid SQL-scripts for integration test data seed!**

    - Let's check examples of java default style, groovy builders and factories

    ```
    Examples can be found here
    fi.bytecraft.spock.animals.KennelBuilders
    ```

Few tips for idiomatic Groovy in testing

- No semicolons needed
- No return keyword needed
- Create new list
  - `["item1", "item"]`
- Create map
  - `[key: "value", key2: value]`
  - `Empty: [:]`
- Instead of setters and getters, use **direct accessors**
  - `myObject.getSomething() → myObject.something`
  - `myObject.setSomething("value") → myObject.something = "value"`
- Use **types** in method signatures
- Prefer **type inference** in variable assignment
  - `MyObject myObject = new MyObject() → def myObject = new MyObject()`
- You can omit the parenthesis for top level expressions
  - `thrown(RuntimeException.class) → thrown RuntimeException`
- Truthy & falsy values
  - `myObject != null → myObject`
  - `myObject == null → !myObject`
- String interpolation
  - `"The value is " + value + " currently" → "The value is $value currently"`

Most important techniques for self-documenting, readable tests

- Extract method
- Name things descriptively
- No magic variables
- The closer you can create the test context to the test method, the better
- Tie your assertions against the created context objects
- Use the correct gherkin-blocks
- Use commented labels in gherkin-blocks

First, let’s check example how not to do it

`Example specification can be found here`
`fi.bytecraft.spock.animals.KennelSpec`

Spock 101: More exercises

1. **SPECIFICATION STYLE:** Refactor `fi.bytecraft.spock.animals.KennelSpec` for self-documenting tests, aka specifications
2. **TEST DATA BUILDING:** Refactor the integration test with sql test seed to be created with objects + repository: `fi.bytecraft.spock.reviewservice.AddReactionISqlSpec`
3. **ISOLATION:** Implement unit test for SSO service successful login, check also that the UserLoginInfo gets the right info for persistence
4. Check the Spock BDD reports produced under *build/spock-reports/index.html*

References

[1] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesøy, B. Helmkamp, and D. North, The RSpec Book: Behaviour-driven Development with RSpec, Cucumber, and Friends. Pragmatic Bookshelf Series, Pragmatic Bookshelf, 2010.

[2] R. Osherove, The Art of Unit Testing, Second Edition. Manning Publications Company, 2013.

[3] J. A. Whittaker, "What is software testing? and why is it so hard?," IEEE software, vol. 17, no. 1, pp. 70–79, 2000.

[4] L. Prechelt, H. Schmeisky, and F. Zieris, "Quality experience: a grounded theory of successful agile projects without dedicated testers," in Proceedings of the 38th International Conference on Software Engineering, pp. 1017–1027, ACM, 2016.

[5] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in Proceedings of the 7th International Workshop on Automation of Software Test, pp. 36–42, IEEE Press, 2012.

[6] L. Williams, G. Kudrjavets, and N. Nagappan, "On the effectiveness of unit test automation at microsoft.," in ISSRE, pp. 81–89, 2009. [27] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 571–579, IEEE, 2005.

[7] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 571–579, IEEE, 2005.

[8] J. Langr, A. Hunt, and D. Thomas, Pragmatic Unit Testing in Java 8 with JUnit. Pragmatic Bookshelf, 2015.

[9] K. Kapelonis, Java Testing with Spock. Manning Publications Company, 2016.

[10] P. Runeson, "A survey of unit testing practices," IEEE software, vol. 23, no. 4, pp. 22–29, 2006.

[11] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on, pp. 201–211, IEEE, 2014.

[12] P. Runeson, "A survey of unit testing practices," IEEE software, vol. 23, no. 4, pp. 22–29, 2006.

[13] L. Williams, G. Kudrjavets, and N. Nagappan, "On the effectiveness of unit test automation at microsoft.," in ISSRE, pp. 81–89, 2009.

[14] B. Li, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and N. A. Kraft, "Automatically documenting unit test cases," in Software Testing, Verification and Validation (ICST), 2016 IEEE International Conference on, pp. 341–352, IEEE, 2016.