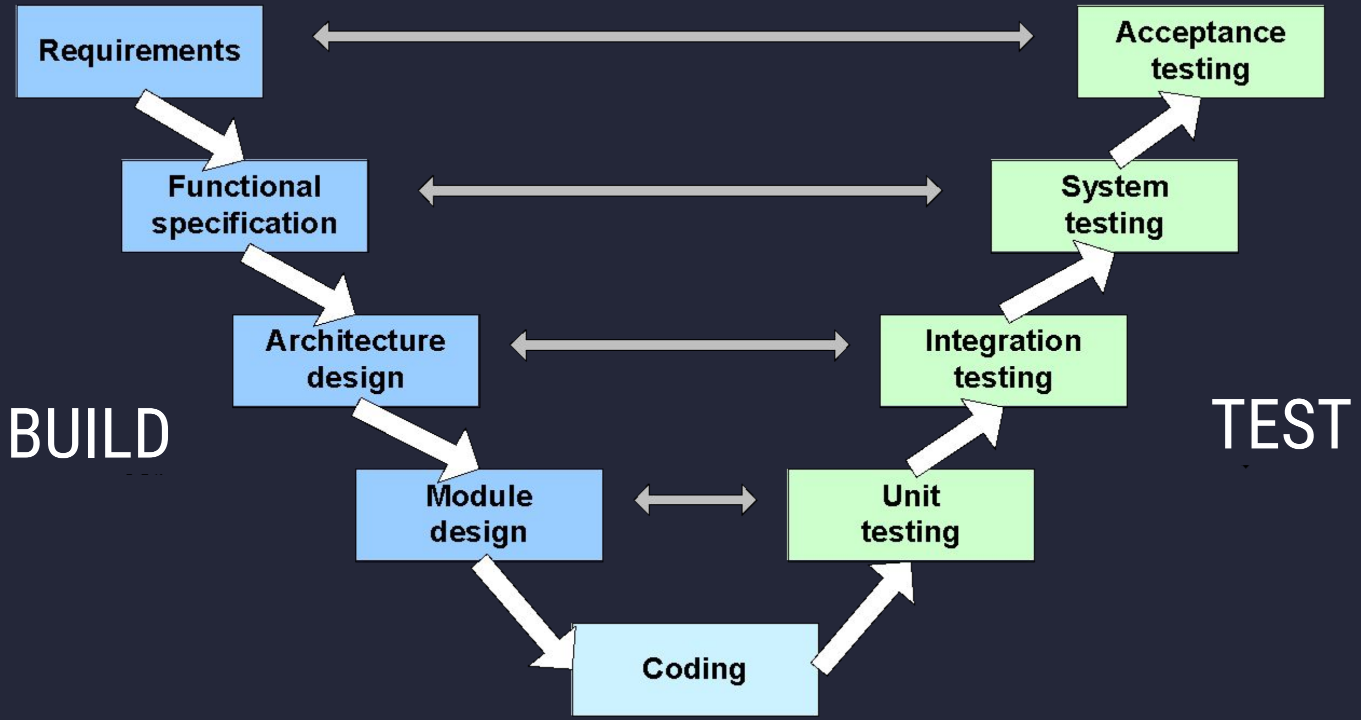
Automated unit & integration testing

By Bytecraft_

Automated Testing

- Testing levels
- Research & Motivation
- Unit testing
- Integration testing
- Integration vs Unit testing
- Frontend: React examples
- Discussions

Testing levels



Automated testing: Research and motivation

Benefits

- Rapid feedback [4]
- Improved product quality [5, 6]
- Increased test coverage [5]
- Increased developer confidence [5]
- Reduced testing time [5]
- Shorter release cycle [7]
- Increased testability design [7]
- Act as documentation [1, 8, 9]
- Continuous regression [6]

Drawbacks

- Can’t replace manual testing [5]
- Maintaining difficulty [5, 10]
- Lack of skilled people [5, 10]
- Hard to select correct testing strategies [5, 7]
- Brittle tests [7]
- More development time [6]
- Cost versus value [10]
- Unmaintained tests can lose all value [7]

Unit testing

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

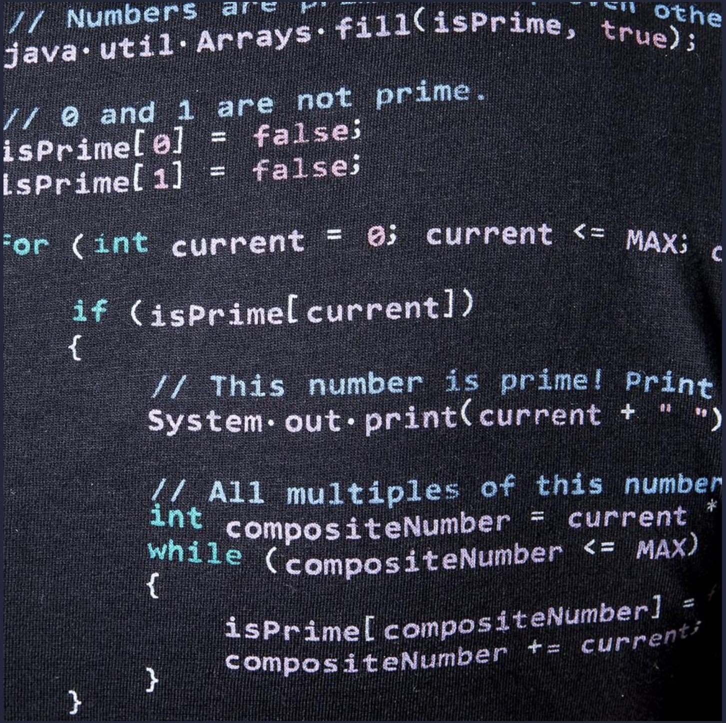
- A good unit test is [3]
  - maintainable
  - readable
  - isolated
  - single concern
  - minimal amount of repetition

Unit testing - JUnit: simple example

- Adding tests to existing method
  - Gathering test coverage
    - Testing exceptions

code:https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/main/java/fi/aalto/testingandqa/algorithm/CurlyBracesChecker.java

a bad test:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/algorithm/BadCurlyBracesCheckerTest.java



```
// Numbers are [illegible] othe
java.util.Arrays.fill(isPrime, true);

// 0 and 1 are not prime.
isPrime[0] = false;
isPrime[1] = false;

for (int current = 0; current <= MAX; c

    if (isPrime[current])
    {
        // This number is prime! Print
        System.out.print(current + " ")

        // All multiples of this number
        int compositeNumber = current *
        while (compositeNumber <= MAX)
        {
            isPrime[compositeNumber] =
            compositeNumber += current;
        }
}
```

What is good testing?

- Inherent role of automated testing is to **verify**
- But in can also **document** from whole feature requirements to single functions
- Code coverage: https://en.wikipedia.org/wiki/Code_coverage
  - How many production code lines are covered by the test suite
  - How does code coverage relate to automated testing roles?

Unit test - verifying and documenting

- Refactoring a poorly documenting Python PYTest
- Maintainability:
  - Removing repetition
    - Using fixture methods
    - Using helper methods
- Readability
  - Separating concerns
  - Naming things
  - Get rid of magic constants
  - Creating your own test DSL

source-code:
https://github.com/anttiahonen/python-unit-testing-example/tree/master/example
test source-code:
https://github.com/anttiahonen/python-unit-testing-example/tree/master/example/tests
(files without the word _commented_)
commented test source-code:
https://github.com/anttiahonen/python-unit-testing-example/tree/master/example/tests
(files with the word _commented_)

Integration testing

- Testing activity which involves multiple components [2, 3]
- Testing a unit of work with real dependencies in place [2]:
  - Database
  - networking etc...
- Not as fast as unit testing
  - Context loading is slow, for example dependency injection containers such as Spring Framework

Integration testing
JUnit: SpringBoot example

- Context loading
- Testing with in-memory db
- Let's do some refactoring

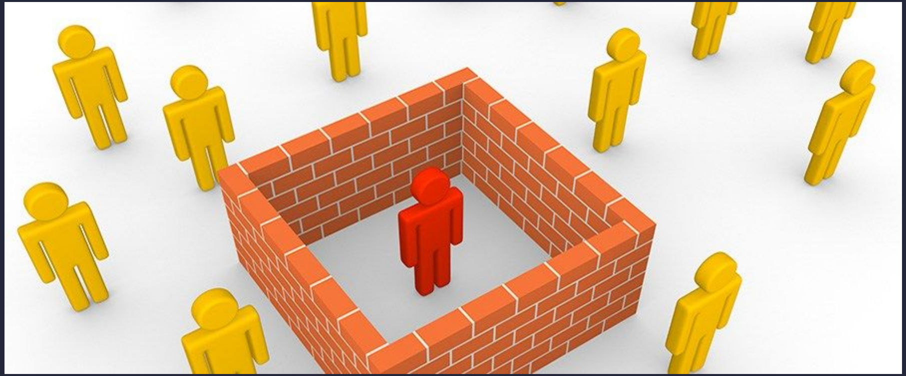
source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/tree/master/src/main/java/fi/aalto/testingandqa/review
(ReviewService.java addComment is the top class / method under test)

test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/reviewservice/AddCommentITest.java
and:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/ReviewServiceBase.java

commented test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/reviewservice/CommentedAddCommentITest.java

Integration testing vs Unit testing

- **Isolation** is the key difference
  - In unit tests, scope can be a lot smaller
- Speed is the second big noticeable difference



Isolation

- **Mocking**: substituting real objects with limited functionality provided by mocks
- **Stubbing**: injecting outputs for mocked object behaviors

Isolation provides

- Determinism
- Enables TDD/BDD

Unit vs. Integration testing mocking & stubbing examples

JUnit with Mockito

source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/tree/master/src/main/java/fi/aalto/testingandqa/review
(ReviewService.java addComment is the top class / method under test)

test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/reviewservice/AddCommentTest.java
and:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/ReviewServiceBase.java
commented test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/java/fi/aalto/testingandqa/review/reviewservice/CommentedAddCommentTest.java

Spock

source-code: still the same ReviewService.addComment

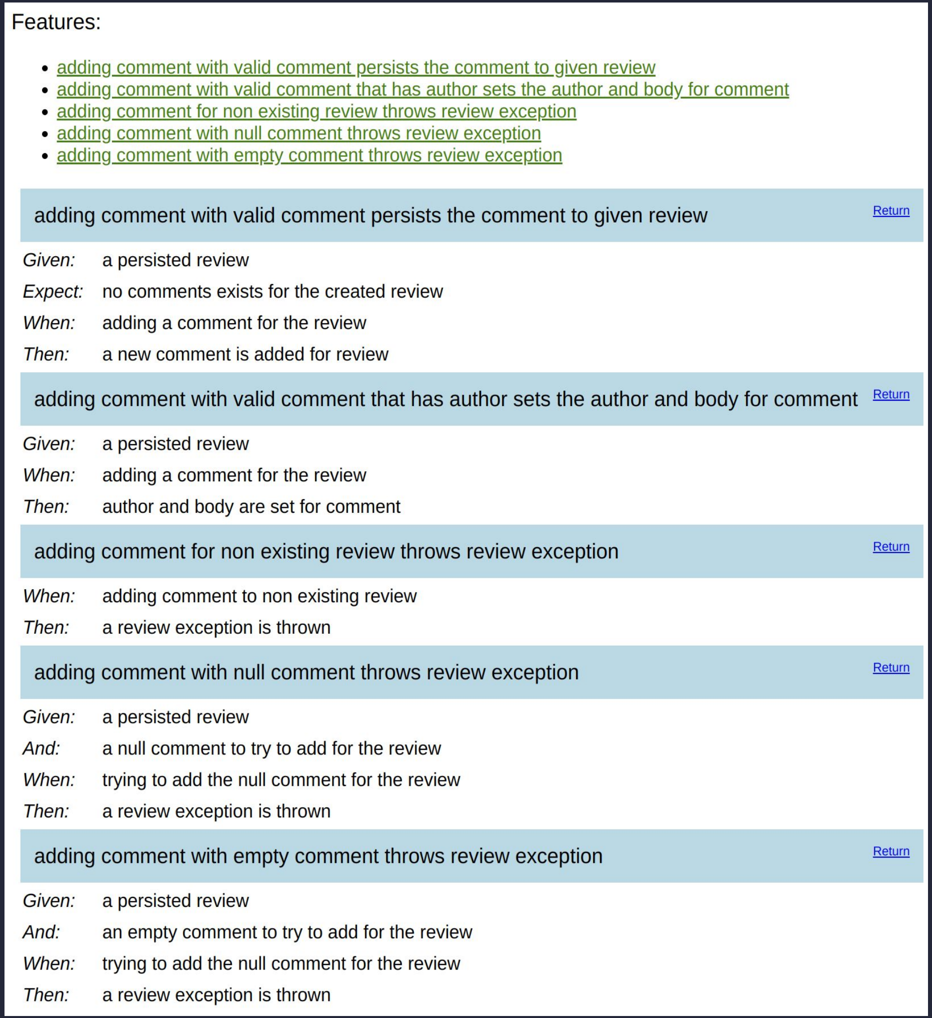
test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/groovy/fi/aalto/testingandqa/reviewservice/AddCommentSpec.groovy

commented test source-code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/groovy/fi/aalto/testingandqa/reviewservice/CommentedAddCommentSpec.groovy

Throwback to good comments

Structured comments that generate **living documentation,** example from **Spock**

Test source code:
https://github.com/anttiahonen/junit-spock-testing-examples/blob/master/src/test/groovy/fi/aalto/testingandqa/reviewservice/AddCommentISpec.groovy



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

Frontend: React

- Test framework is **Jest**
  - Spec-style (also has support for traditional xUnit-style)
- **React testing library** is the “official” way (of create-react-app template) to do React testing
  - Philosophy is to do assertions against what is visible on the screen
    → Try to avoid testing DOM internals, such as does element have classes or id

source-code:
https://github.com/anttiahonen/react-testing-library-examples/tree/master/src (foods/Foods.js is component under test)
test source-code:
https://github.com/anttiahonen/react-testing-library-examples/blob/master/src/foods/Foods.spec.js

Check these for more info how to use Spec-style keywords for self-documenting tests:
start:https://github.com/anttiahonen/ekanban/blob/master/frontend/src/tests/unit/components/Game.bad.spec.js
better:https://github.com/anttiahonen/ekanban/blob/master/frontend/src/tests/unit/components/Game.better.spec.js
best-with-comments:https://github.com/anttiahonen/ekanban/blob/master/frontend/src/tests/unit/components/Game.best.withcomments.spec.js

Discussions

- What kind of testing have you thought of using in the project?
- What is the role of automation?
- Any complex testing needs that don’t directly fit in functional testing of single service?
- Testing of quality attributes?
- How will you use the PO for testing?

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.