Unit test problems – how not to die of old age and stay calm

Mateusz Winnicki

www.mateuszwinnicki.pl
mateusz.winnicki@euvic.pl

Unit test – so hard to define

Why should I even care?

(https://www.computer.org/csdl/mags/so/2007/03/s3024.html)

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect	Quality effect
Sanchez et al.[6]	Case study	5 years	Yes	IBM	Point-of-sale device driver	Medium	9–17	Java	Increased effort 19%	40%†
Bhat and Nagappan[7]	Case study	4 months	No	Microsoft	Windows networking common library	Small	6	C/C++	Increased effort 25–35%	62%†
	Case study	≈7 months	No	Microsoft	MSN Web services	Medium	5–8	C++/C#	Increased effort 15%	76%†
Canfora et al.[8]	Controlled experiment	5 hours	No	Soluziona Software Factory	Text analyzer	Very small	28	Java	Increased effort by 65%	Inconclusive based on quality of test
Damm and Lundberg[9]	Multi-case study	1–1.5 years	Yes	Ericsson	Components for a mobile network operator application	Medium	100	C++/Java	Total project cost increased by 5–6%	5–30% decrease in fault-slip-through rate; 55% decrease in avoidable fault costs
Melis et al.[10]	Simulation	49 days (simulated)	No	Calibrated using Klondike-Team and Quinary data	Market information project	Medium	4‡	Smalltalk	Increased effort 17%	36% reduction in residual defect density
Mann[11]	Case study	8 months	Yes	PetroSleuth	Windows-based oil and gas project management with statistical modeling elements	Medium	4–7	C#	n/a	81%§; customer and developers' perception of improved quality
Geras et al.[12]	Quasi-controlled experiment	3 hours	No	Various companies	Simple database-backed business information system	Small	14	Java	No effect	Inconclusive based on failure rates; improved based on no. of tests and frequency of execution
George and Williams[13]	Quasi-controlled experiment	4.75 hours	No	John Deere, Role Model Software, Ericsson	Bowling game	Very small	24	Java	Increased effort 16%	18%#
Ynchausti[14]	Case study	8.5 hours	No	Monster Consulting	Coding exercises	Small	5	n/a	Increased effort 60–100%	38–267%†

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect	Quality effect
Flohr and Schneider[15]	Quasi-controlled experiment	40 hours	Yes	University of Hannover	Graphical workflow library	Small	18	Java	Improved productivity by 27%	Inconclusive
Abrahamsson et al.[16]	Case study	30 days	No	VTT	Mobile application for global markets	Small	4	Java	Increased effort by 0% (iteration 5) to 30% (iteration 1)	No value perceived by developers
Erdogmus et al.[17]	Controlled experiment	13 hours	No	Politecnico di Torino	Bowling game	Very small	24	Java	Improved normalized productivity by 22%	No difference
Madeyski[18]	Quasi-controlled experiment	12 hours	No	Wroclaw University of Technology	Accounting application	Small	188	Java	n/a	–25 to –45%†
Melnik and Maurer[19]	Multi-case study	4-month projects over 3 years	No	University of Calgary/ SAIT Polytechnic	Various Web-based systems (surveying, event scheduling, price consolidation, travel mapping)	Small	240	Java	n/a	73% of respondents perceive TDD improves quality
Edwards[20]	Artifact analysis	2–3 weeks	No	Virginia Tech	CS1 programming assignment	Very small	118	Java	Increased effort 90%	45%†
Pančur et al.[21]	Controlled experiment	4.5 months	No	University of Ljubljana	4 programming assignments	Very small	38	Java	n/a	No difference
George[22]	Quasi-controlled experiment	1-3/4 hours	No	North Carolina State University	Bowling game	Very small	138	Java	Increased effort 16%	16%†
Müller and Hagner[23]	Quasi-controlled experiment	10 hours	No	University of Karlsruhe	Graph library	Very small	19	Java	No effect	No effect, but better reuse and improved program understanding

If it is so helpful why I don’t like it?

Test name

Patterns

- methodName_stateUnderTest_expectedBehavior
- methodName_expectedBehavior_stateUnderTest
- test[feature being tested] - testIfFailToWithdrawMoneyIfAccountIsInDebt)
- feature beign tested - shouldFailIfTryToWithdrawMoneyFromAccountInDebt
- should_expectedBehavior_when_stateUnderTest
- given_preconditions_when_stateUnderTest_then_expectedBehavior
- when_stateUnderTest_expect_expectedBehavior

Tests are written AFTER implementation

Downsides

- **no influence on a design**
- zero help during implementation
- no fast feedback
- slowing down the implementation
- not all requirements are tested
- writing tests just to pass
- “is it hard to write? Just skip”

Use TDD/BDD always when you are writing new code

Too many assertions

Too many tests in one

```java
@Test
public void calculateByStringCalculator() {
    final StringCalculator calc = new StringCalculator();

    final int result1 = calc.calculate( s: "2 + 2");
    assertEquals( expected: 4, result1);
    final int result2 = calc.calculate( s: "5 + 7");
    assertEquals( expected: 12, result2);
    final int result3 = calc.calculate( s: "10 - 2");
    assertEquals( expected: 8, result3);
}
```

```java
@Test
public void calculate_sumTwoNumbers() {
    final StringCalculator calc = new StringCalculator();

    final int result = calc.calculate( s: "2 + 2");
    assertEquals( expected: 4, result);
}

@Test
public void calculate_sumThreeNumbers() {
    final StringCalculator calc = new StringCalculator();

    final int result = calc.calculate( s: "5 + 7");
    assertEquals( expected: 13, result);
}

@Test
public void calculate_subtractTwoNumbers() {
    final StringCalculator calc = new StringCalculator();

    final int result = calc.calculate( s: "10 - 2");
    assertEquals( expected: 8, result);
}
```

Too many assertions

```java
@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_foundOnlyFromRequestedCompany() {
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));

    final String requestedCompany = "EUVIC";
    final List<Person> allFromCompany = personService.findAllFromCompany(requestedCompany);

    assertEquals( expected: 2, allFromCompany.size());
    assertTrue(allFromCompany.stream().map(Person::getCompany).allMatch(c -> c.equals(requestedCompany)));
    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Daniel")));
    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Mateusz")));
}
```

```java
@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_allPeopleInResponseShareRequestedCompany() {
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));

    final String requestedCompany = "EUVIC";
    final List<Person> allFromCompany = personService.findAllFromCompany(requestedCompany);

    assertTrue(allFromCompany.stream().map(Person::getCompany).allMatch(c -> c.equals(requestedCompany)));
}

@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_twoPeopleAreReturned() {
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));

    final String requestedCompany = "EUVIC";
    final List<Person> allFromCompany = personService.findAllFromCompany(requestedCompany);

    assertEquals( expected: 2, allFromCompany.size());
}

@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_twoSpecificPeopleFromDatabaseAreReturned() {
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));

    final String requestedCompany = "EUVIC";
    final List<Person> allFromCompany = personService.findAllFromCompany(requestedCompany);

    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Daniel")));
    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Mateusz")));
}
```

Dependencies between tests

Order?!

```java
@Test
@TestOrder(1)
public void addPerson() {}

@Test
@TestOrder(2)
public void fetchPerson() {}

@Test
@TestOrder(3)
public void deletePerson() {}
```

Tests have to be independent

Independent to extreme level

```java
private DatabaseConnection databaseConnection;
private PersonService personService;
private static final String REQUESTED_COMPANY = "EUVIC";

@Before
public void setup() {
    databaseConnection = setupDatabase(Mode.H2);
    personService = new PersonService(databaseConnection);

    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));
}

@After
public void tearDown() {
    cleanDatabase(databaseConnection);
}

@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_allPeopleInResponseShareRequestedCompany() {
    final List<Person> allFromCompany = personService.findAllFromCompany(REQUESTED_COMPANY);
    assertTrue(allFromCompany.stream().map(Person::getCompany).allMatch(c -> c.equals(REQUESTED_COMPANY)));
}

@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_twoPeopleAreReturned() {
    final List<Person> allFromCompany = personService.findAllFromCompany(REQUESTED_COMPANY);
    assertEquals( expected: 2, allFromCompany.size());
}

@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_twoSpecificPeopleFromDatabaseAreReturned() {
    final List<Person> allFromCompany = personService.findAllFromCompany(REQUESTED_COMPANY);
    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Daniel")));
    assertTrue(allFromCompany.stream().map(Person::getName).anyMatch(n -> n.equals("Mateusz")));
}
```

Downsides

- big coupling between tests
- you cannot read tests from up to bottom
- one change during test refactoring will blow up all the others
- you cannot introduce new specific tests without changing rest of them

What we can do?

- always generate random literals, numbers etc. which are not important for current test
- setups and cleanups in test method
- Test Data Builders
- **you have to be sane here. In example - setting up a database will be always the same (so it can be in separate method)**

```java
@Test
public void findAllFromCompany_twoInRequestedCompanyOneFromAnother_allPeopleInResponseShareRequestedCompany() {
    final DatabaseConnection databaseConnection = setupDatabase(Mode.H2);
    final PersonService personService = new PersonService(databaseConnection);
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Daniel",  age: 28,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 26,  euvic: "OTHER"));

    final String requestedCompany = "EUVIC";
    final List<Person> allFromCompany = personService.findAllFromCompany(requestedCompany);
    assertTrue(allFromCompany.stream().map(Person::getCompany).allMatch(c -> c.equals(requestedCompany)));

    cleanDatabase(databaseConnection);
}
```

DRY is not really a rule in unit testing

Non-deterministic tests

Generation of something in the production code

```
public Person getRandomPerson() {
    final Random random = new Random();
    final int number = random.nextInt();
    if(number % 2 == 0) {
        return something;
    }
    return somethingElse;
}
```

Try instead:

```
public Person getRandomPerson() {
    final int number = randomProvider.getRandomInt();
    if(number % 2 == 0) {
        return something;
    }
    return somethingElse;
}
```

Randomness in tests:

```java
@Test
public void findAllFromCompany_allBeforeNow() {
    final DatabaseConnection databaseConnection = setupDatabase(Mode.H2);
    final PersonService personService = new PersonService(databaseConnection);
    personService.addPerson(new Person( name: "Mateusz",  age: 26,  company: "EUVIC"));
    personService.addPerson(new Person( name: "Daniel",  age: 28,  company: "EUVIC"));
    personService.addPerson(new Person( name: "Pawel",  age: 26,  company: "OTHER"));

    final Instant now = Instant.now();
    final List<Person> response = personService.findAllCreatedBefore(now);
    assertEquals( expected: 3, response.size());

    cleanDatabase(databaseConnection);
}
```

Try instead:

```java
@Test
public void findAllFromCompany_allBeforeNow() {
    final DatabaseConnection databaseConnection = setupDatabase(Mode.H2);
    final TimeService timeService = mock(TimeService.class);
    final PersonService personService = new PersonService(databaseConnection, timeService);
    when(timeService.getNow()).thenReturn(Instant.ofEpochMilli(0L));

    personService.addPerson(new Person( name: "Mateusz",  age: 26,  company: "EUVIC"));
    personService.addPerson(new Person( name: "Daniel",  age: 28,  company: "EUVIC"));
    personService.addPerson(new Person( name: "Pawel",  age: 26,  company: "OTHER"));

    final Instant now = Instant.ofEpochMilli(10L);
    final List<Person> response = personService.findAllCreatedBefore(now);
    assertEquals( expected: 3, response.size());

    cleanDatabase(databaseConnection);
}
```

Slow unit test

IO operations

```
@Before
public void setup() {
    databaseConnection = setupDatabase(Mode.H2);
    personService = new PersonService(databaseConnection);
}

@After
public void tearDown() {
    cleanDatabase(databaseConnection);
}

@Test
public void calculateAverageAgeInCompany_threePeopleInDatabaseFromEuvic_averageAgeIsCalculatedOnlyForEuvic() {
    databaseConnection.execute(insertQuery( name: "Mateusz",  age: 26,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Kamil",  age: 25,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Witold",  age: 27,  euvic: "EUVIC"));
    databaseConnection.execute(insertQuery( name: "Pawel",  age: 30,  euvic: "OTHER"));

    double averageAge = personService.calculateAverageAgeInCompany( euvic: "EUVIC");

    assertEquals( expected: 26.0d, averageAge,  delta: 0.01d);
}
```

Split the IO and algorithm

```
public class People {

    private final List<Person> people;

    public People(final List<Person> people) {
        this.people = people;
    }

    OptionalDouble calculateAverageAgeInCompany(final String companyName) {
        return people.stream()
            .filter(p -> companyName.equals(p.getCompany()))
            .mapToInt(Person::getAge)
            .average();
    }
}
```

Use mocks

```
@Before
public void setup() {
    databaseConnection = mockDatabase(Mode.H2);
    personService = new PersonService(databaseConnection);
}

@After
public void tearDown() {
    cleanMock(databaseConnection);
}

@Test
public void calculateAverageAgeInCompany_threePeopleInDatabaseFromEuvic_averageAgeIsCalculatedOnlyForEuvic() {
    when(databaseConnection.findAll()).thenReturn(Arrays.asList(
        new Person( name: "Mateusz",  age: 26,  company: "EUVIC"),
        new Person( name: "Kamil",  age: 25,  company: "EUVIC"),
        new Person( name: "Witold",  age: 27,  company: "EUVIC"),
        new Person( name: "Pawel",  age: 30,  company: "OTHER")
    ));

    final double averageAge = personService.calculateAverageAgeInCompany( euvic: "EUVIC")
        .orElseThrow(() -> new IllegalArgumentException("Should be present here!"));

    assertEquals( expected: 26.0d, averageAge,  delta: 0.01d);
}
```

Intentional waiting

```
@Test
public void findNotificationsForUserId_paginationFirstPageSizeOne_checkTotalSize() throws IOException {
    final String userId1 = "USER1";
    final NotificationDocument nd1 = new NotificationDocument()
            .timestamp(Instant.ofEpochMilli(50L)).recipient(new Recipient().userId(userId1));
    final NotificationDocument nd2 = new NotificationDocument()
            .timestamp(Instant.ofEpochMilli(51L)).recipient(new Recipient().userId(userId1));
    client.getHighLevelClient().index(new IndexRequest(indexTestName, "notification").id("1").source(mapper.writeValueAsString(nd1), XContentType.JSON)
    client.getHighLevelClient().index(new IndexRequest(indexTestName, "notification").id("2").source(mapper.writeValueAsString(nd2), XContentType.JSON)
    elasticSearchPropagationWait(2000L);

    final PageResponse<NotificationDocument> page = notificationRepository.findNotificationsForUserId(userId1, 0, 1);

    assertEquals(2L, page.getTotalCount().longValue());
}
```

Blocking API

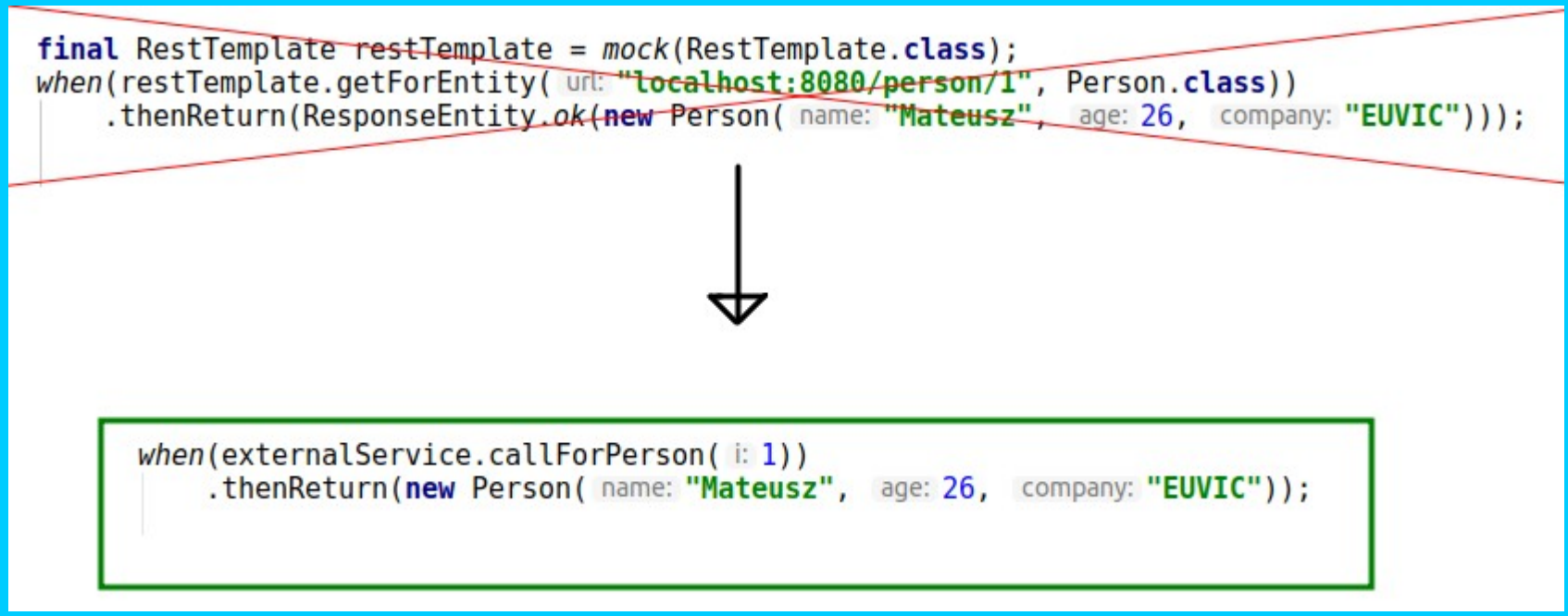
```
@Test
public void findNotificationsForUserId_paginationFirstPageSizeOne_checkTotalSize() throws IOException {
    final UUID userId1 = UUID.fromString("00000000-0000-0000-0000-000000000001");
    final String nd1 = "{\"recipient\":{\"user-id\":\"" + userId1 +"\"},\"@timestamp\":\"2018-01-31T09:39:34.740Z\"}";
    final String nd2 = "{\"recipient\":{\"user-id\":\"" + userId1 +"\"},\"@timestamp\":\"2018-01-31T09:39:35.740Z\"}";
    final BulkRequest request = new BulkRequest().setRefreshPolicy("wait_for");
    request.add(new IndexRequest(indexTestName, "notification").id("1").source(nd1, XContentType.JSON));
    request.add(new IndexRequest(indexTestName, "notification").id("2").source(nd2, XContentType.JSON));
    client.getHighLevelClient().bulk(request);

    final PageResponse<JsonNode> page = notificationRepository.findNotificationsForUserId(userId1, 0, 1);

    assertEquals(2L, page.getTotalElements().longValue());
}
```

Problem with mocks

Do not mock what you don’t own



Do not mock if you don’t have to

```
when(databaseConnection.findAll()).thenReturn(Arrays.asList(
    new Person( name: "Mateusz",  age: 26,  company: "EUVIC"),
    new Person( name: "Kamil",  age: 25,  company: "EUVIC")
));

// ********************************************************************* //

final PersonRepository mockedPersonRepository = mock(PersonRepository.class);
when(mockedPersonRepository.findAll()).thenReturn(Arrays.asList(
    new Person( name: "Mateusz",  age: 26,  company: "EUVIC"),
    new Person( name: "Kamil",  age: 25,  company: "EUVIC")
));

// ********************************************************************* //

final PersonRepository personRepository = new InMemoryPersonRepository(); //HashMap implementation
personRepository.store(new Person( name: "Mateusz",  age: 26,  company: "EUVIC"));
personRepository.store(new Person( name: "Kamil",  age: 25,  company: "EUVIC"));
```

In memory real implementations

Increasing coverage to infinity

Should I?

- makes no sense to test getters/setters
- some tests will mock the whole world and test nothing
- test coverage doesn’t tell you anything about the quality of your tests

What can happen?

- testing directly dumb methods
- you will hack your production code to create test (changing private to higher visibility). **And I am not talking about code refactoring.**
- you will start using reflection in you tests

Testing API of external libraries or generated code

Core rules!

- **always use TDD/BDD**
- too many assertions are not fine
- staying DRY generally will not help
- unit test has to be fast
- don’t mock too much
- 100% is not your ultimate goal

Q&A