Lesson 04
Test Management 1/2

DEVOPS - ITI 4
HEI 2020-2021

Why testing is important ?

To err is human, but to really foul things up you need a computer
Paul R. Elrich

Previously, in 1996

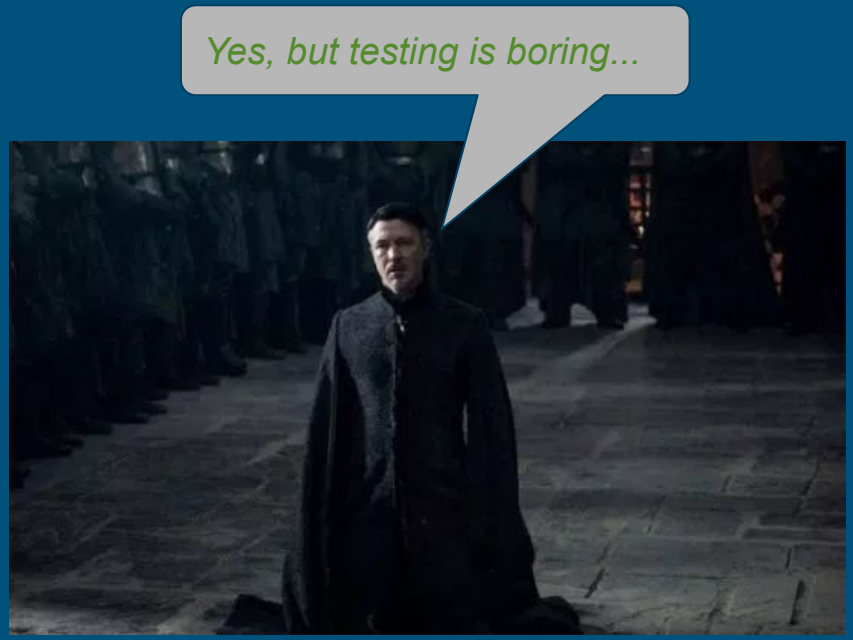
Ariane V fail :

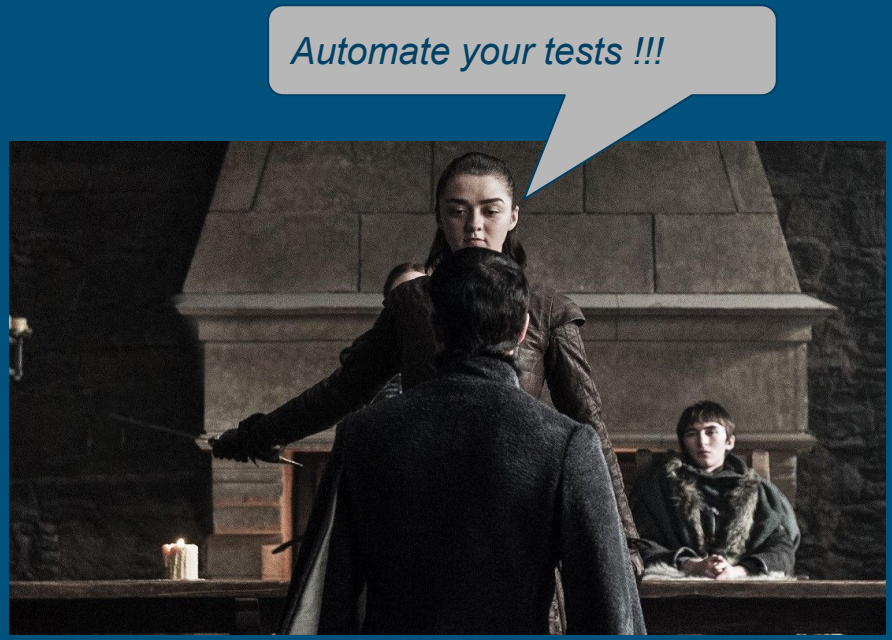
- Root cause : Integer overflow in automatic pilot module
- Cost ~ 370 M €
- In order to economise 120k €, CNES don’t test this module

Tests goals :

- To detect issues before production
- To ensure the proper working of all the functionalities
- To be sure that the system is reliable

Stereotype





How should we test ?

Program testing can be used to show the presence of bugs, but never to show their absence !
Edsger Dijkstra

Check that item :

- gives expected outputs when inputs are valid
- does not give expected outputs when inputs are invalid
- should never crash

Testing levels

Overview

Unit Testing	*To check each component (unit of source code) individually.*
Integration testing	*To check the interconnection between the different components.*
System testing	*In a complete and integrated system to verify its compliance with specified requirements..*
Operational acceptance testing	*To verify that the product conforms to the specification*

Example

- Unit testing :
  - Could I open the drawer ?
- Integration testing :
  - Could I open the drawers ?
- System testing :
  - Could I open the drawer 1000 times ?
- Operational acceptance testing :
  - Could I use the drawer ?

Way to write tests

GWT

Describe the initial context of the system.

Describe an event/action.

Describe an expected outcome/result.

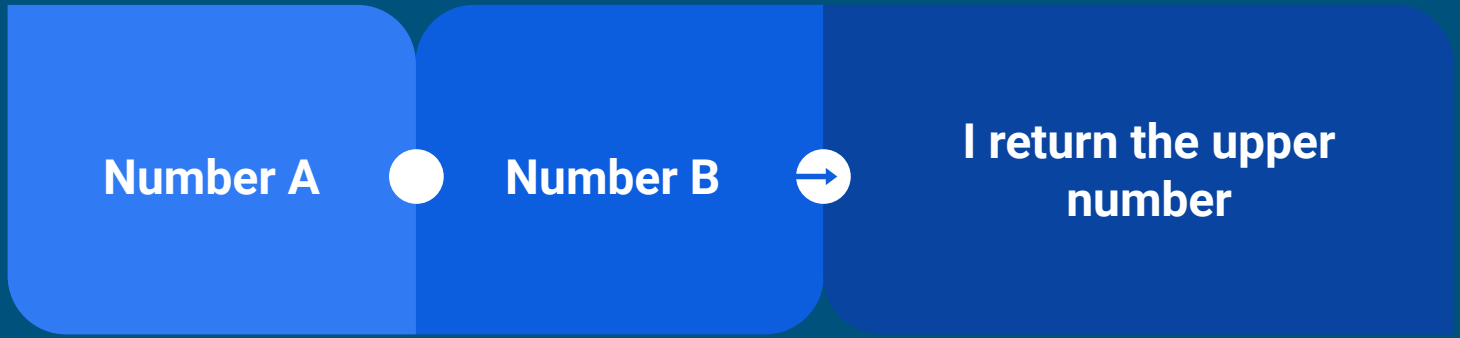
Example

GIVEN
There are 4 beers

WHEN
We drink 3 beers

THEN
It should have 1 beer

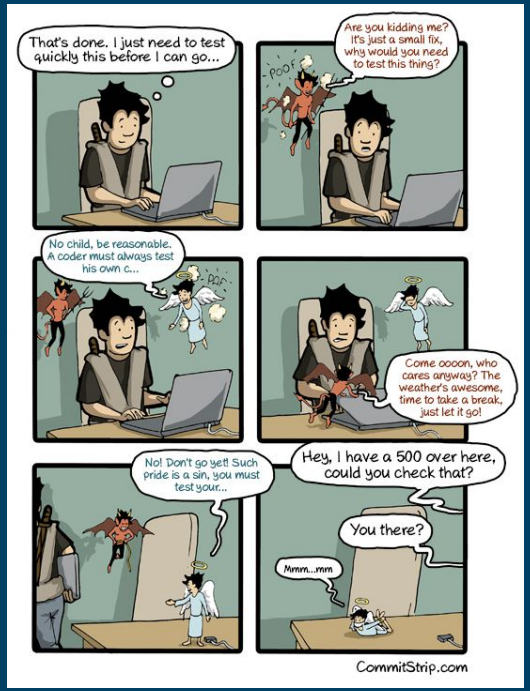
What are tests for this function ?



Solution

	A greater than B	A less than B	A equal to B
Given	*A = 2 and B = 1*	*A = 1 and B = 2*	*A = 1 and B = 1*
When	*Call the function with A and B*		
Then	*Return A*	*Return B*	*Return A or B*

How build a test with JUnit ?



JUnit

Java platform Unit testing framework

Using JUnit

- Library => not in the JDK

- To add the library to the project with maven :

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
</dependency>
```

Test case

- Is a java class that regroups the tests for a component

- Contains several tests

Write a test method

Test

- is a *public* method in a test case
- Have *@Test* annotation
- Return *void*

Naming convention/habit

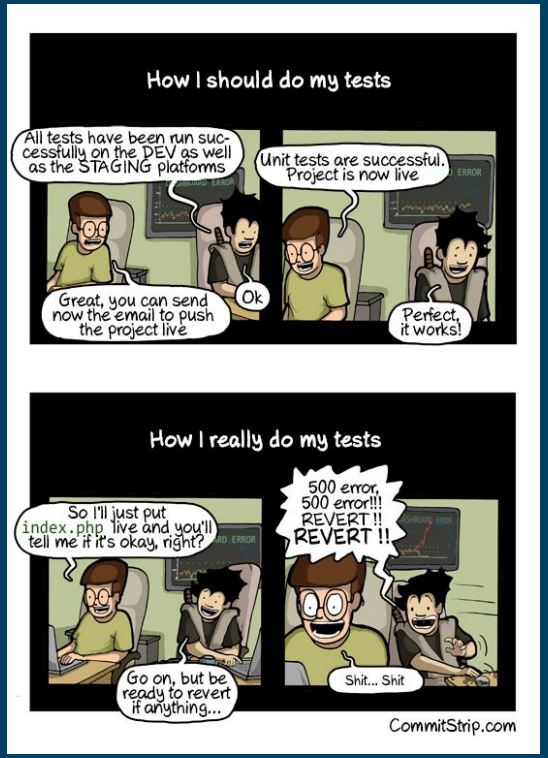
- Your test will be more readable if the name of your test is explicit

- Method’s name must start with “should” :
  - *shouldReturnTrue()*
  - *shouldThrowNotFoundException()*
  - *shouldDivideAndReturnResult()*

Example

```
@Test
public void shouldAddAndReturnResult() {
        //Given
        int addend1=1;
        int addend2=5;
        int sum=6;

        //When
        int result = operationService.add(addend1, addend2);

        //Then
        ...
}
```

How check result with JUnit ?



Result of a test

- Success : All assertions are correct
- Failure : At least one assertion is incorrect
- Error : An unexpected exception has been thrown

Test result of a method

Assertions

- Use to check result of a test during THEN step

- *Assert* class contains static method to do assertions :
  - *assertEquals(expected, actual) ;*
  - *assertNotEquals(expected, actual);*
  - *assertNull(object);*
  - *assertNotNull(object);*
  - *assertTrue(condition);*
  - *assertFalse(condition);*

Example

```
@Test
public void shouldAddAndReturnResult() {
        //Given
        int addend1=1;
        int addend2=5;
        int sum=6;

        //When
        int result = operationService.add(addend1, addend2);

        //Then
        assertEquals(sum,result);
}
```

Test Exception

Why do we have to test the exceptions ?

An exception to handle a non-standard return of a method

Test the exceptions

- Check the throw of a correct exception when the variables are not correct.

- Check the throw of a correct exception when the method have an external problem.

Specific check for an exception

- Used to fail a test when an expected exception has not been thrown :
  - *fail(reason) ;*
  - *fail();*

- Used to catch an expected exception :
  - *expected* attribute in *@Test* annotation

Example

```
    @Test(expected = DivisionByZeroException.class)
    public void shouldDivideAndThrowDivisionByZeroException() throws
DivisionByZeroException {
        //Given
        int numerator=16;
        int denominator=0;

        //When
        operationService.divide(numerator, denominator);

        //Then
        fail("Should throw a DivisionByZeroException");
    }
```

How to manage data for tests ?

Scenario

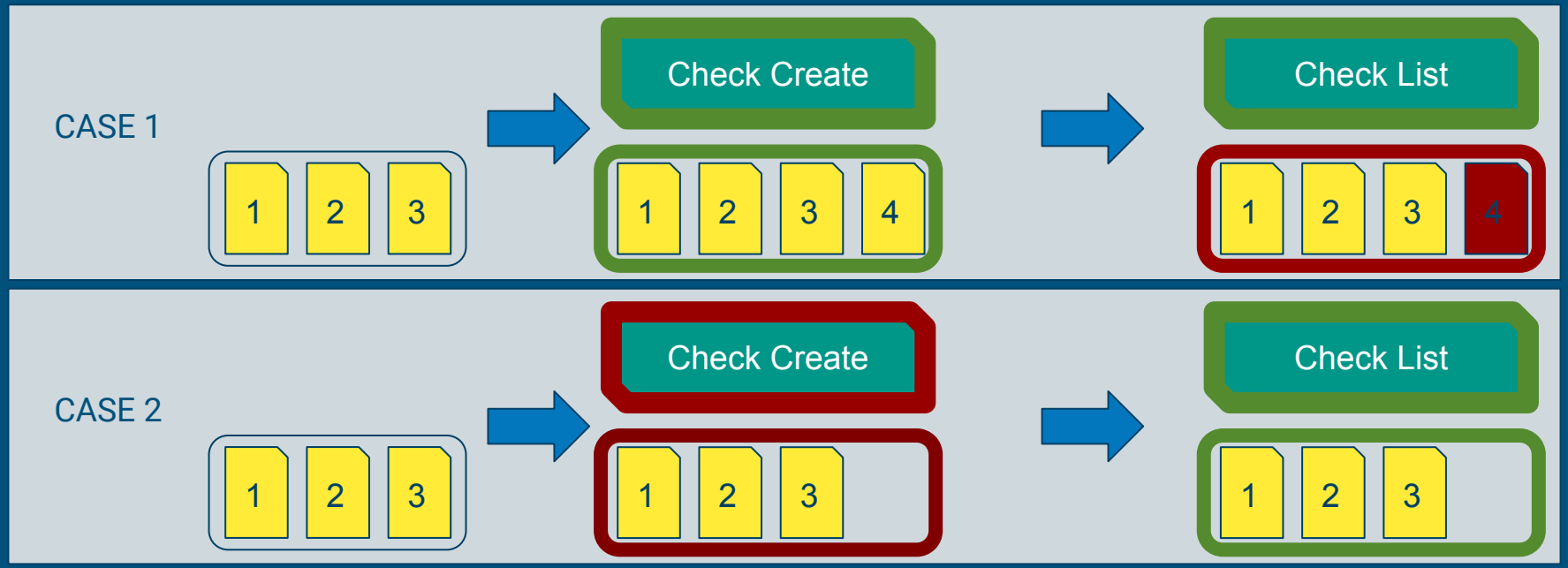
Context

- We have a service which manage files :
  - createFile : Create a file in a directory
  - listFiles : List files in the directory
- We want to check these methods

TestCase - Build

	Check Create	Check List
GIVEN	• Path to the directory • Name to the new file • Number of file in the directory	• Path to the directory • Number of file in the directory
WHEN	*Call the method*	
THEN	• A new file exist in the directory with the correct name • The number of files has been increased by 1	• The number of files is the same

TestCase - Run



The result of the test of the method list depends entirely on the result of the test of the method Create !!!

What we need

- Init data before each test

- Purge data after each test

JUnit tools

@Before and @After

- *@Before* annotates a method that will be executed before each test

- *@After* annotates a method that will be executed after each test

@BeforeClass and @AfterClass

- *@BeforeClass* annotates a method that will be executed once before all the tests of a test case.

- *@AfterClass* annotates a method that will be executed once after all the tests of a test case.

These methods must be static

Example 1/2

```
private final static String pathExampleRepository ="src/test/resources/FileServiceTestCase";

private static File exampleRepository;
private static File tmpRepository;

private FileService fileService;

@BeforeClass
public static void runOnceBeforeClass() throws IOException {
        System.out.println("@BeforeClass - runOnceBeforeClass");

        tmpRepository = Files.createTempDirectory(Paths.get("target"),"tmp-").toFile();
        exampleRepository = new File(pathExampleRepository);
}

@AfterClass
public static void runOnceAfterClass() {
        System.out.println("@AfterClass - runOnceAfterClass");
        tmpRepository.delete();
}
```

Example 2/2

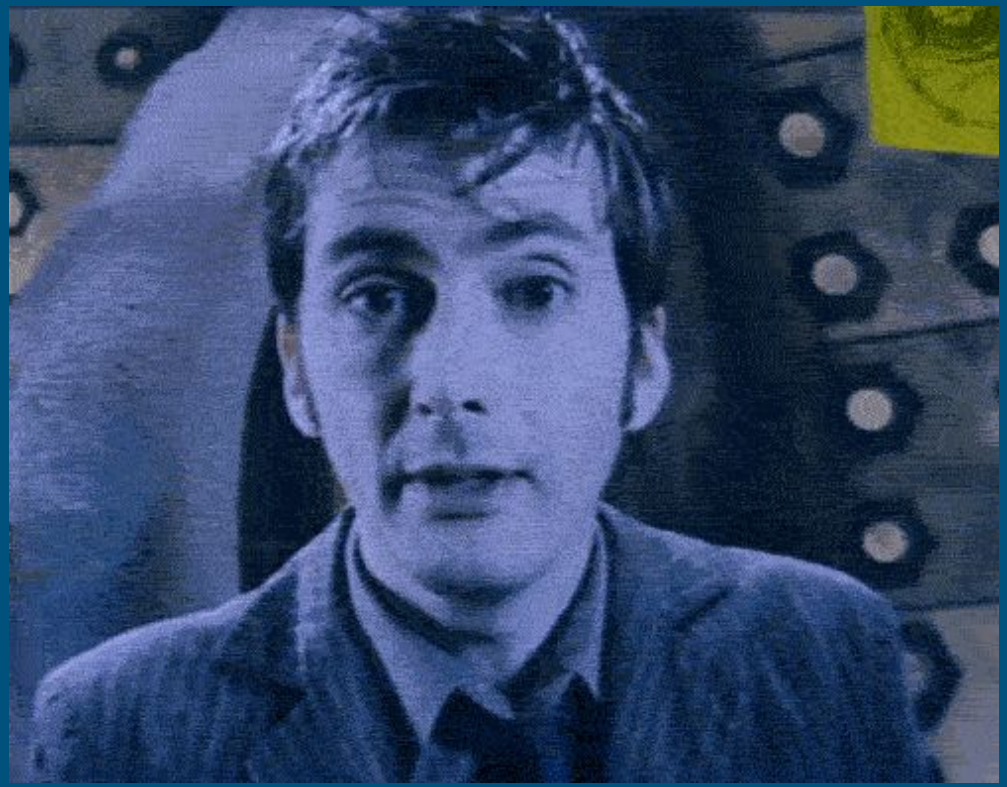
```
@Before
public void runBeforeTestMethod() throws IOException {
        System.out.println("@Before - runBeforeTestMethod");
        fileService = new FileServiceImpl(tmpRepository);

        for(File sourceFile  : exampleRepository.listFiles()) {
                Files.copy(sourceFile.toPath(), new File(tmpRepository,sourceFile.getName()).toPath());
        }
}

@After
public void runAfterTestMethod() {
        System.out.println("@After - runAfterTestMethod");
        for(File file  : tmpRepository.listFiles()) {
                file.delete();
        }
}
```

Question

What tests are needed?

```
    public void deleteFile(String name) throws
FileNotFoundException {
        File file = new File(repository, name);
        if(file.exists()) {
            file.delete();
        }else {
            throw new FileNotFoundException(name);
        }
    }
```

Thank you for your attention !



Links :

- Sources
  - http://www.commitstrip.com/
  - https://giphy.com/
  - https://www.jmdoudoux.fr/java/dej/chap-junit.htm
  - https://en.wikipedia.org/wiki/Software_testing
  - https://junit.org/junit4/
  - https://www.mkyong.com/unittest/junit-4-tutorial-1-basic-usage/

- Examples :
  - https://gitlab.com/hei-devops/lesson/lesson-2021/lesson-04