



Unit Testing

Write code to test the code
you just wrote

**When I join a new Dev-Team
and want to write Unit-Tests**



"We don't do that here"



@kode.on.kanvas

#10xCoding



About this presentation

- What you will find
 - Definition & benefits of Unit Testing
 - A little coding example to get the feeling of Unit Testing using JUnit and Mockito in spring boot 2
- What you won't find
 - An in-depth explanations of above mentioned frameworks



Agenda

- Introduction
 - Without Unit Testing
 - What is Unit Testing
 - What Unit Testing is not
 - Benefits
 - Rumors & answers to them
 - Best Practices and Guidelines
- Practical Example in Java
 - Junit + Mockito + spring boot



Programmer's Life Without Unit Testing

- Writes a lot of code.
- Someday something stops working
- Fixing a bug is easy
- Hard part is finding it (read, Spending hours in front of debugger)
- Fixing the bug can break other parts of application
- Bugs re-appear mysteriously



What is Unit Testing

- A **unit test** is an automated and self-checked procedure used to validate that a particular **unit/module** of source code works as expected.
 - A module is typically a Class
- Write tests for all methods of the module
- The class under test should be tested in isolation



What is it not

- A test is not a unit test if -
 - It talks to database
 - It communicates across the network
 - It touches the file system
 - It can't run at the same time as any of your other unit tests
 - You are not testing the class in isolation



Benefits of Unit Testing

- Strict, written contract that the piece of code must satisfy
- Allows us to refactor code and make sure that it still works correctly
- Simplifies Integration -
 - Testing the parts of program first and then testing their integration
- Provides Documentation
 - Devs can look at the unit test to understand how to use the module



Benefits of Unit Testing

- Improves code quality
 - Code that cannot be easily tested is not factored properly
- Separation of Interface from implementation
 - A class may have many collaborating classes
 - Mock Objects act as an interface for those classes
- Loosely coupled code
- Minimizing dependencies



Rumors

- *“It requires time and I am always overscheduled”*
- *“It’s a waste of time.: If I change my classes, I will have to rewrite the tests!”*
- *“My code is rock solid. I don’t need tests!”*



Answers

- *I am in a hurry* issue leads to a vicious cycle.
 - More pressure you feel -> less tests you write
 - Less tests you write -> less productive you are and the less stable your code becomes
 - The less productive and accurate you are -> more pressure you feel




Answers

- **Unit Testing speeds up your development**
 - Automated and self-checked code reduces the time to manually debug your code
 - Writing tests immediately after you code new features(or even better, *before*) lets you find bugs sooner and isolated(easier to correct)
- **Unit Testing lets you refactor safely**
 - A change in code that causes a regression is found immediately



Refactoring

- Definition
 - A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour - Martin Fowler
- Unit Testing and Refactoring go hand in hand
 - Before refactoring a section of code you better have solid tests that cover it
 - Test, small change, test, small change, test,...



Experience is a hard teacher because she
gives the test first, the lesson afterward.

- A Chinese Proverb



Best Practices / Guidelines

- Don't test Java, PHP, javascript
- Don't test POJO objects
- Mock all external services and states
- Test one code unit at a time
- Write multiple test scenarios
- Unit tests should be isolatable
- Unit tests should be repeatable
- Unit tests should be easy to write



Design is not what is looks like and feels like.
Design is how it works.

- Steve jobs



Best Practices/Guidelines

- If the tests get complicated, we need better code design
- How to find bad code design -
 - Complex test objects need to be created to test simple scenarios (otherwise null pointers are fired)
 - Lot of dependencies need to be mocked
 - Too many mock statement for one test scenario
 - Test is getting more than ~5-10 lines of code
 - Refactor your code for any of the above scenarios



Best Practices/Guidelines

- Write a test for every bug you find.
- How will this help us -
 - Will keep an eye on bug for us
 - Will prevent us from making changes that will make the bug reappear
 - It is the agile way
 - Is is always good to have more unit tests
- Unit test should be fully automated and non-interactive



Best Practices/Guidelines

- Measure the tests
 - Use code coverage tools to understand what parts of your code are not tested
- Fix Failing tests immediately
- Keep testing at unit level
 - Unit testing is about testing classes.
 - Do not test the entire workflow
- Keep tests independent



Best Practices/Guidelines

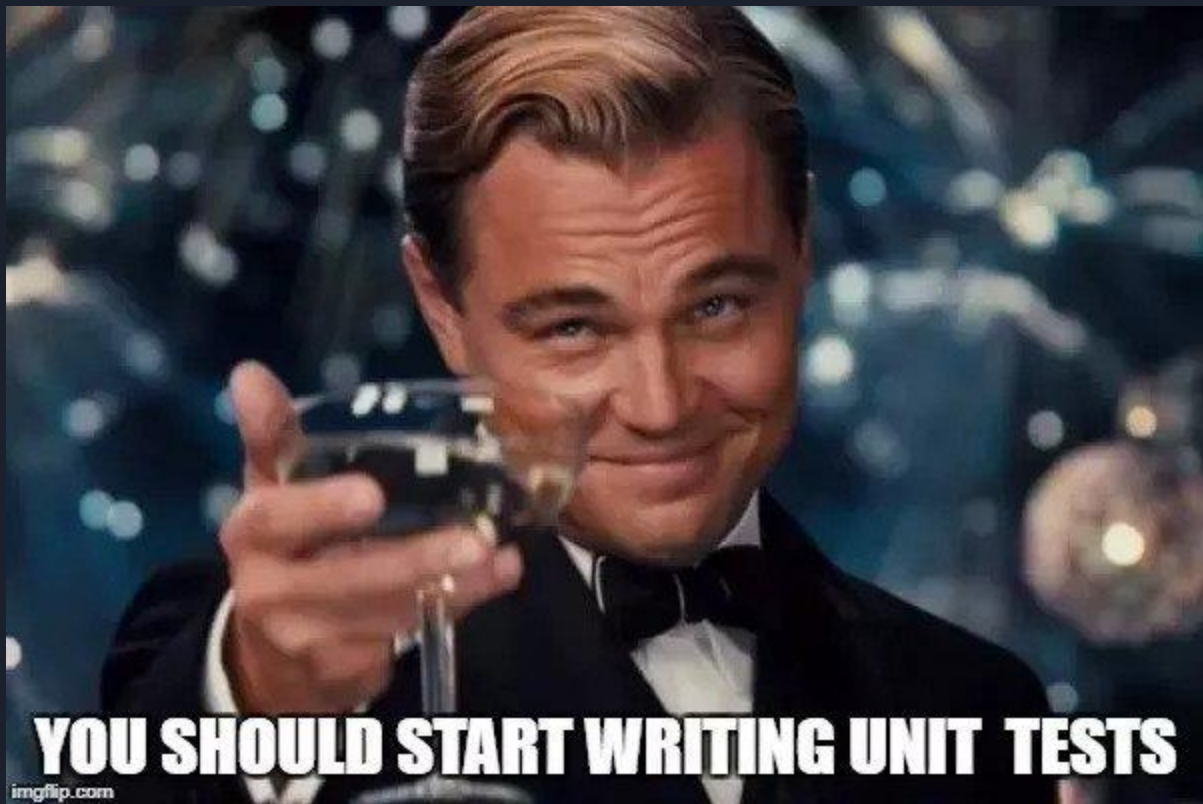
- Cover boundary cases - NaN, null, negative integers
- Test with randomly generated parameters
- Test each feature once
- Provide Negative tests
- Design code with testing in mind



Sample Example - Junit, mockito, spring boot 2.0



Questions?



YOU SHOULD START WRITING UNIT TESTS

imgflip.com



Thank you