

Java

Les 9

jeroen.devos01@ap.be



AP HOGESCHOOL
ANTWERPEN

AP.BE

Lesdoelen

- Unit testing
- Test Driven Development
- Unit testing met AI



Advanced Java™



Dev Team



QA

Automatische testen

1. Graphical User Interface (GUI) testen

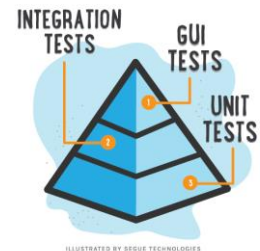
- Gebruik van een toepassing opnemen of programmeren
- GUI test tool genereert input en clicks in de toepassing
- GUI test tool kijkt de output na

2. Integration testen

- Test de interactie tussen verschillende onderdelen (layers) van een toepassing
- Test de 'happy flow'

3. Unit testen

- Test een individueel onderdeel van een toepassing : één class of één method
- Test alle gedrag van het onderdeel



Unit test

- Test één onderdeel
- Weinig testcode
- Snelle test
- Test een verwacht resultaat
- Test staat op zichzelf / geïsoleerd:
 - alle data nodig voor de test wordt bij start van de test opgebouwd
 - geen afhankelijkheden van andere testen
- Vuistregel : schrijf minstens 2 unit testen per te testen functionaliteit
 - één waarbij het goed loopt
 - één waarbij het fout loopt



JUnit

- is een populair unit testing framework voor Java
 - meer dan 60% van de Java projecten gebruikt JUnit
 - open source
 - gebruikt Java annotations
-
- er zijn uiteraard nog veel meer unit testing frameworks, enkel andere bekende frameworks zijn:
 - TestNG
 - JTest



JUnit
Software

TestNG
Software

Jtest
Software

+ Vergelijking toevoegen

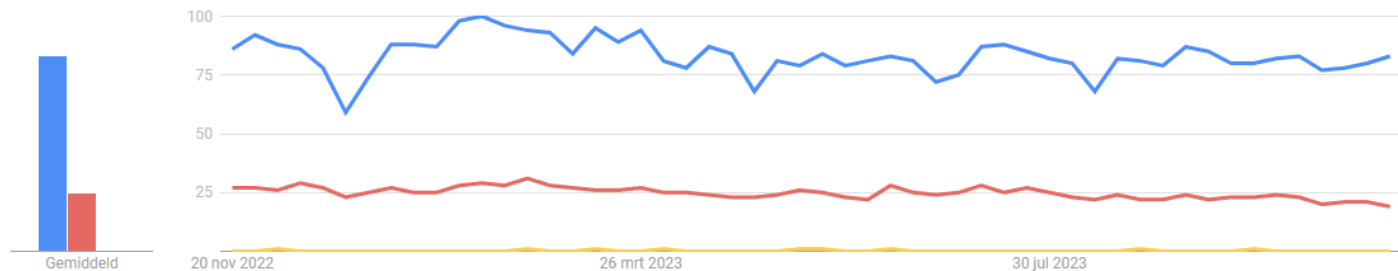
Wereldwijd ▼

Afgelopen 12 maanden ▼

Alle categorieën ▼

Google Zoeken ▼

Interesse in de loop der tijd ?



JUnit maven dependency

```
<dependency>  
  <groupId>org.junit.jupiter</groupId>  
  <artifactId>junit-jupiter</artifactId>  
  <version>5.10.1</version>  
  <scope>test</scope>  
</dependency>
```

JUnit Jupiter versie bevat een test engine om de unit testen uit te voeren.

Een eerste test schrijven

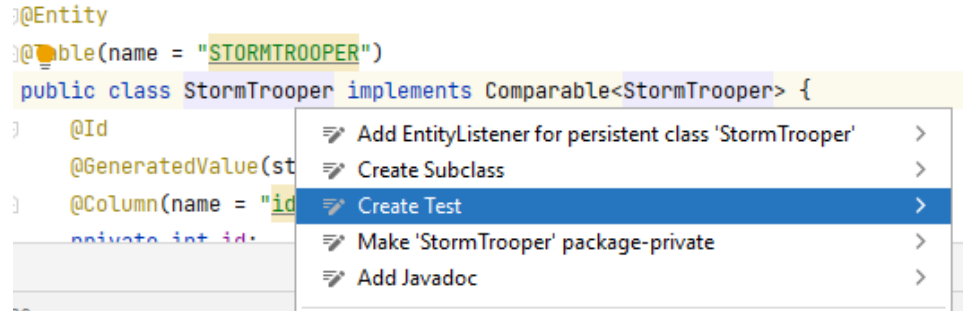
```
class StormTrooperTest {  
    @Test  
    void getName() {  
        String trooperName = "Test";  
        StormTrooper trooper = new StormTrooper(trooperName, null);  
        assertEquals(trooperName, trooper.getName());  
    }  
}
```

Unit test class regels

1. Unit test classes komen in de /test/java folder
2. Unit test classes volgen dezelfde package structuur als de te testen class
3. Unit test class naamgeving = naam v/d te testen class + “Test”

Tip : gebruik IntelliJ !

ALT+ENTER op class naam en dan krijg je de mogelijkheid om een test class aan te maken.



Unit test regels

1. Elke test methode heeft annotation `@Test`
2. Elke test methode staat op zichzelf : de methode heeft geen globale attributen van de class nodig.
3. Elke test methode test één onderdeel : **er moet een assert in staan**

```
@Test
void getName() {
    String trooperName = "Test";
    StormTrooper trooper = new StormTrooper(trooperName, rank: null);
    assertEquals(trooperName, trooper.getName());
}
```

Unit test Assertions

Onderdeel van het JUnit Jupiter framework

‘Assertion’ = stelling, veronderstelling, bewering

Er worden twee waarden aan de veronderstelling meegegeven : de verwachte waarde (expected) en de effectieve waarde (actual):

- `assertEquals(expected, actual)`
- `assertNotEquals(expected, actual)`

Test faalt als assertion niet waar is.

Unit test Assertions

En nog enkele speciale gevallen:

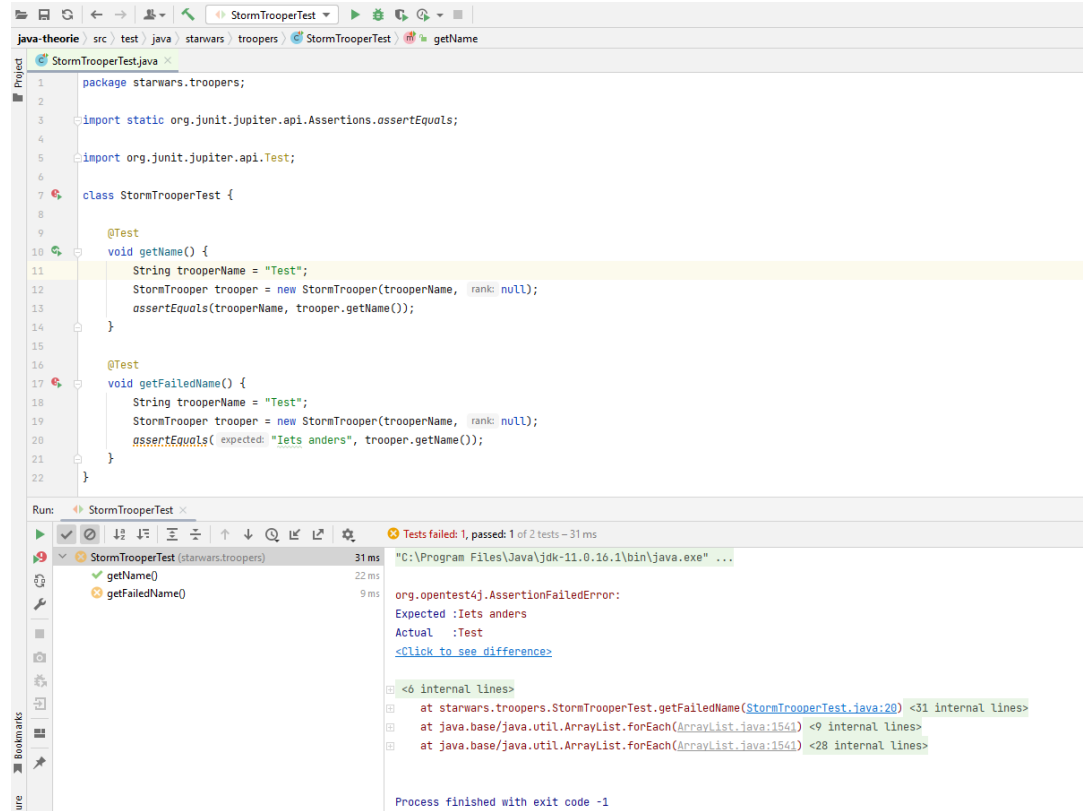
- `assertTrue(actual)`
- `assertFalse(actual)`
- `assertNull(actual)`
- `assertNotNull(actual)`

Unit test Assertions

1 unit test = 1 assertion
(minstens)

Unit test uitvoeren

- Uitvoeren zoals een main method (groene pijltje)
- Test run tab opent onderaan:
 - Opvolging testresultaten
 - Stack trace bij gefaalde test



The screenshot shows an IDE with a Java project named 'java-theorie'. The source code for 'StormTrooperTest.java' is visible, containing two test methods: 'getName()' and 'getFailedName()'. The 'getName()' method passes, while 'getFailedName()' fails due to an assertion error. The 'Run' tab at the bottom displays the test results, showing that 1 test failed and 1 test passed. The stack trace for the failed test is visible, indicating the failure occurred in the 'getFailedName()' method at line 20.

```
package starwars.troopers;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class StormTrooperTest {

    @Test
    void getName() {
        String trooperName = "Test";
        StormTrooper trooper = new StormTrooper(trooperName, rank: null);
        assertEquals(trooperName, trooper.getName());
    }

    @Test
    void getFailedName() {
        String trooperName = "Test";
        StormTrooper trooper = new StormTrooper(trooperName, rank: null);
        assertEquals( expected: "Iets anders", trooper.getName());
    }
}
```

Run: StormTrooperTest

Tests failed: 1, passed: 1 of 2 tests - 31 ms

Test Method	Duration	Result
StormTrooperTest (starwars.troopers)	31 ms	Passed
getName()	22 ms	Passed
getFailedName()	9 ms	Failed

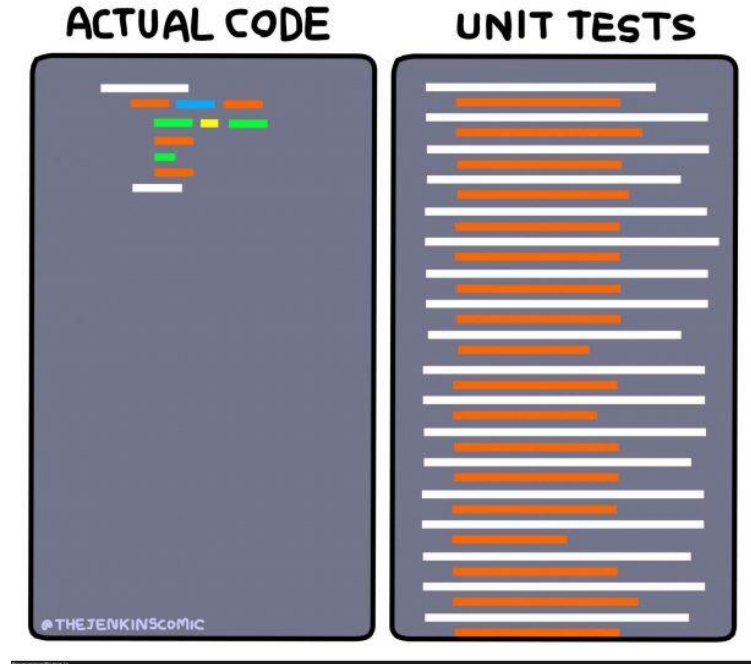
org.opentest4j.AssertionFailedError:
Expected :Iets anders
Actual :Test
[Click to see difference](#)

<6 internal lines>
at starwars.troopers.StormTrooperTest.getFailedName(StormTrooperTest.java:20) <31 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <28 internal lines>

Process finished with exit code -1

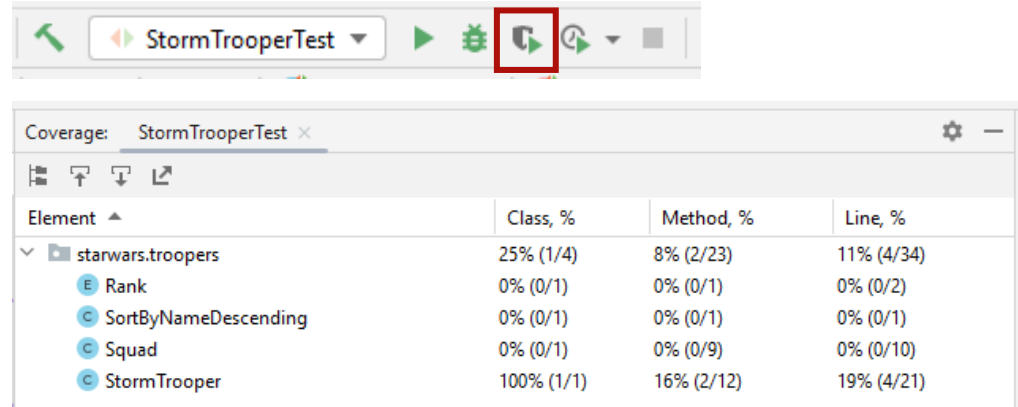
Testen schrijven

- aantal lijnen testcode = 2x tot 3x
aantal lijnen productiecode
- tijd nodig om testen te schrijven =
50% van tijd nodig om feature te
schrijven



Test coverage

- goede coverage = 95% van de lijnen code getest
- IntelliJ – run test coverage



Element	Class, %	Method, %	Line, %
starwars.troopers	25% (1/4)	8% (2/23)	11% (4/34)
Rank	0% (0/1)	0% (0/1)	0% (0/2)
SortByNameDescending	0% (0/1)	0% (0/1)	0% (0/1)
Squad	0% (0/1)	0% (0/9)	0% (0/10)
StormTrooper	100% (1/1)	16% (2/12)	19% (4/21)

Test coverage

Opgelet!
Goede coverage != goede testen

Mocking

Kunnen we een class unit testen als deze andere classes nodig heeft?

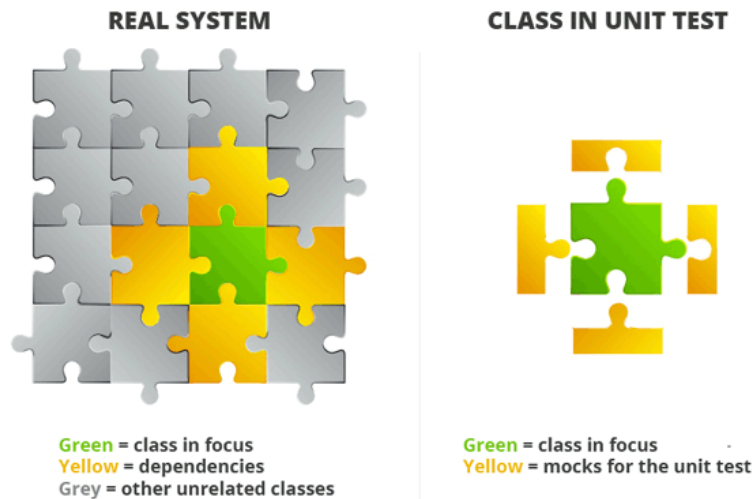
Bv: SquadService heeft de SquadDAO nodig om de methode findAllSquadNames() uit te voeren

```
public class SquadService {  
  
    private final SquadDAO squadDAO;  
    private final StormTrooperDAO stormTrooperDAO;  
  
    public SquadService(SquadDAO squadDAO, StormTrooperDAO stormTrooperDAO) {  
        this.squadDAO = squadDAO;  
        this.stormTrooperDAO = stormTrooperDAO;  
    }  
  
    public List<String> findAllSquadNames() {  
        List<Squad> squads = squadDAO.getSquads();  
        return squads.stream() Stream<Squad>  
            .map(squad -> squad.getName()) Stream<String>  
            .collect(Collectors.toList());  
    }  
}
```

Mocking

Kunnen we een class unit testen als deze andere classes nodig heeft?

→ **Mocking** / “doen alsof”

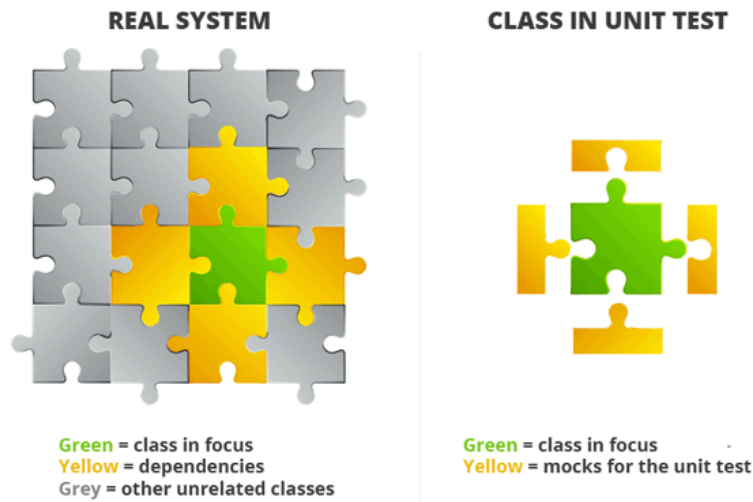


Mocking

Kunnen we een class unit testen als deze andere classes nodig heeft?

→ **Mocking** / “doen alsof”

1. Vervang in de te testen class alle externe dependencies door een ‘mock’.
2. Op de ‘mock’ stel je in welk antwoord deze moet geven als een methode van de class wordt aangesproken.



Mockito

- populair mocking framework
- werkt naadloos samen met JUnit
- andere bekende frameworks:
 - PowerMock
 - EasyMock



● Mockito
Software

● PowerMock
Software

● EasyMock
Software

+ Vergelijking toevoegen

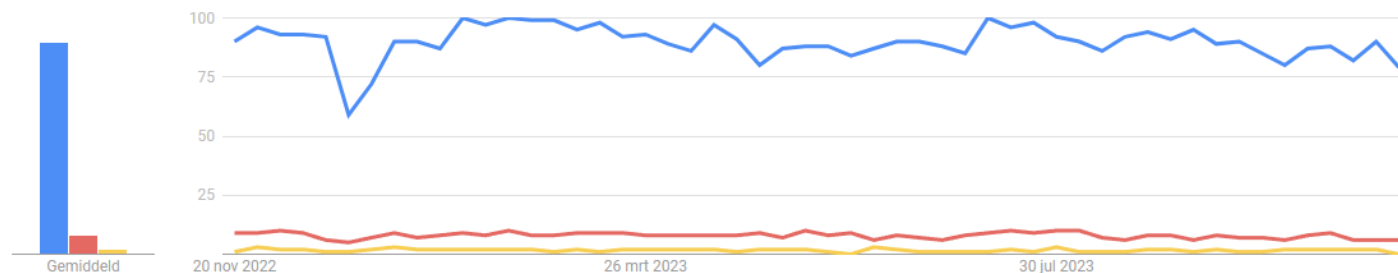
Wereldwijd ▼

Afgelopen 12 maanden ▼

Alle categorieën ▼

Google Zoeken ▼

Interesse in de loop der tijd ?



Mockito maven dependency

```
<dependency>  
  <groupId>org.mockito</groupId>  
  <artifactId>mockito-junit-jupiter</artifactId>  
  <version>5.7.0</version>  
  <scope>test</scope>  
</dependency>
```


Gebruik : when

```
class SquadServiceTest {  
  
    @Test  
    void findAllSquadNames_size2() {  
        SquadDAO squadDAO = mock(SquadDAO.class);  
        SquadService service = new SquadService(squadDAO, null);  
  
        List<Squad> squads = Arrays.asList(new Squad("test1"), new Squad("test2"));  
        when(squadDAO.getSquads()).thenReturn(squads);  
  
        List<String> squadNames = service.findAllSquadNames();  
        assertEquals(2, squadNames.size());  
    }  
}
```

Gebruik : when

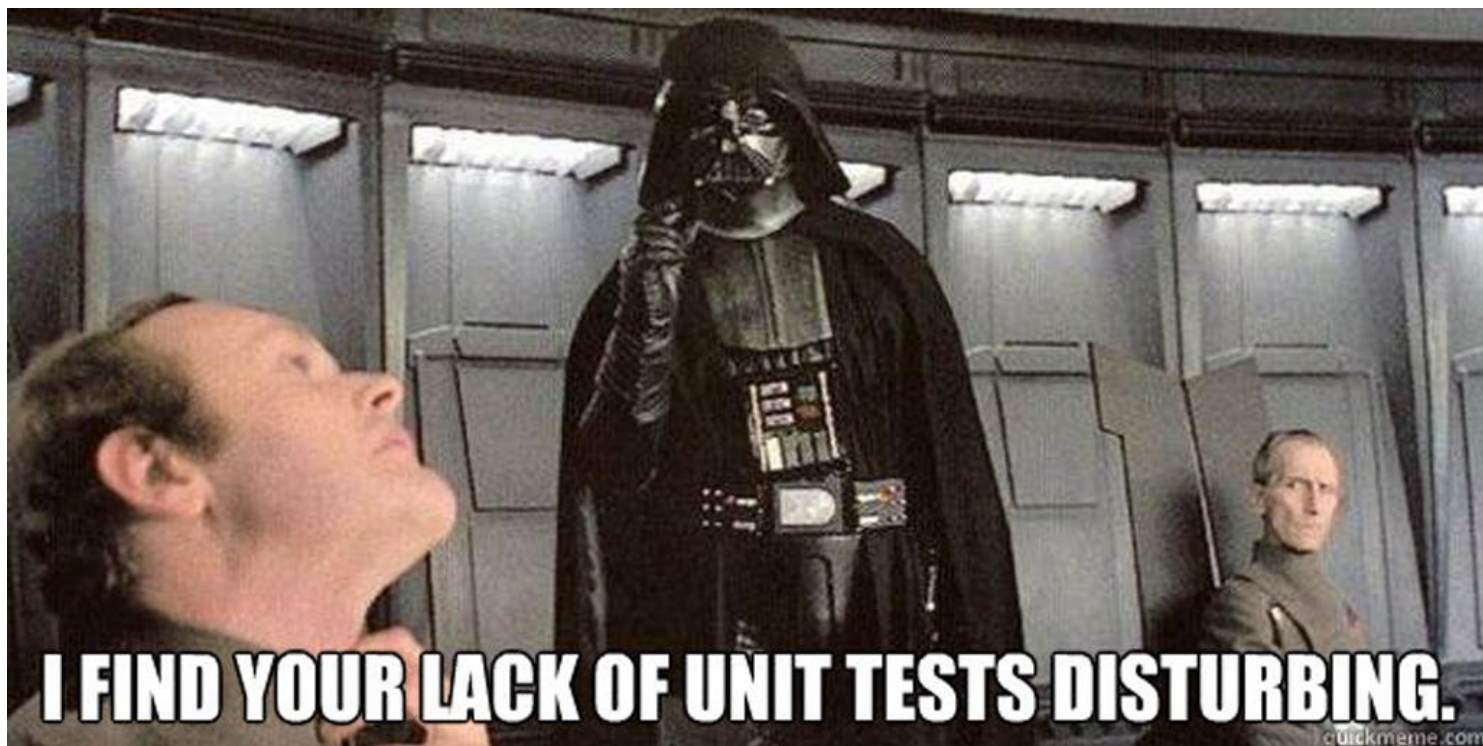
1. mock de SquadDAO :
SquadDAO squadDAO = **mock**(SquadDAO.class);
2. maak een lijst van testgegevens aan
3. de mock geeft de lijst van testgegevens terug:
when(squadDAO.getSquads()).**thenReturn**(squads);

Gebruik : verify

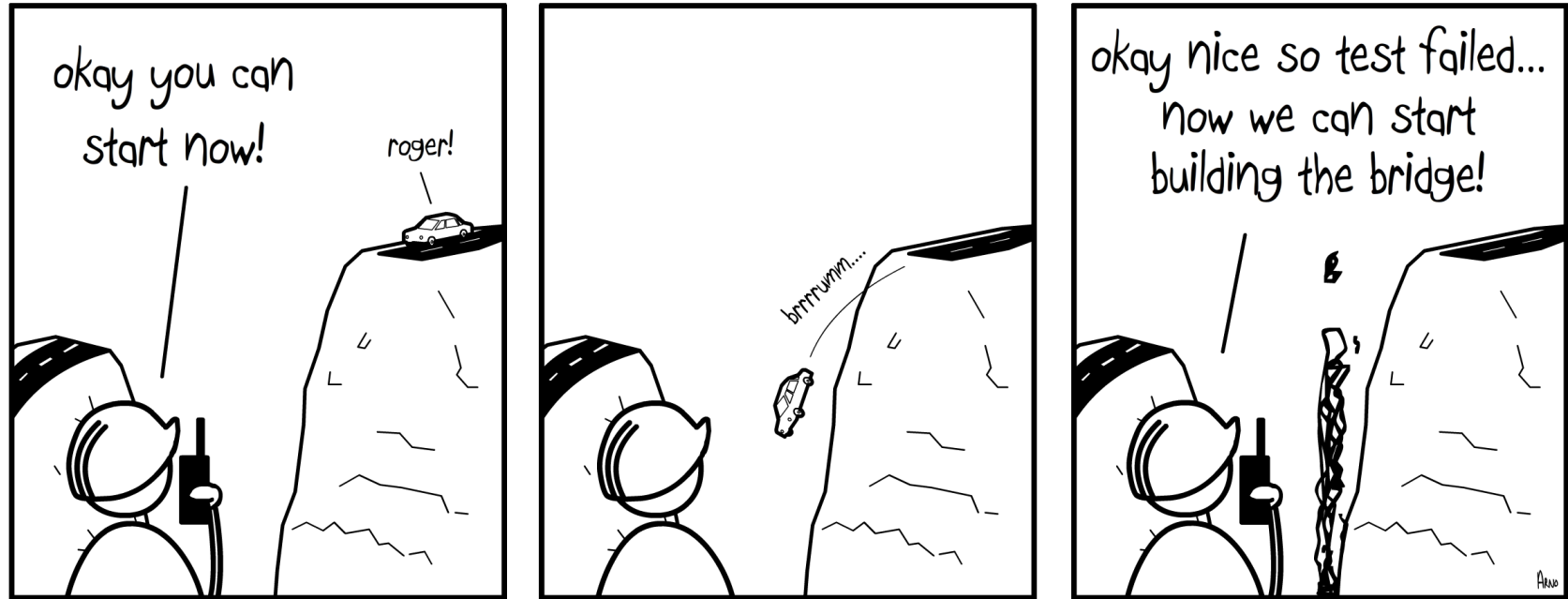
```
class SquadServiceTest {  
  
    @Test  
    void createNewSquad() {  
        SquadDAO squadDAO = mock(SquadDAO.class);  
        SquadService service = new SquadService(squadDAO, null);  
  
        Squad squad = new Squad("test");  
        service.createNewSquad(squad);  
  
        verify(squadDAO, times(1)).createSquad(squad);  
    }  
}
```

Gebruik : verify

1. mock de SquadDAO :
SquadDAO squadDAO = **mock**(SquadDAO.class);
2. voer de actie uit die de mock gebruikt
service.createNewSquad(squad);
3. vraag aan de mock of de methode gebruikt werd, en hoeveel keer:
verify(squadDAO, **times**(1)).createSquad(squad);



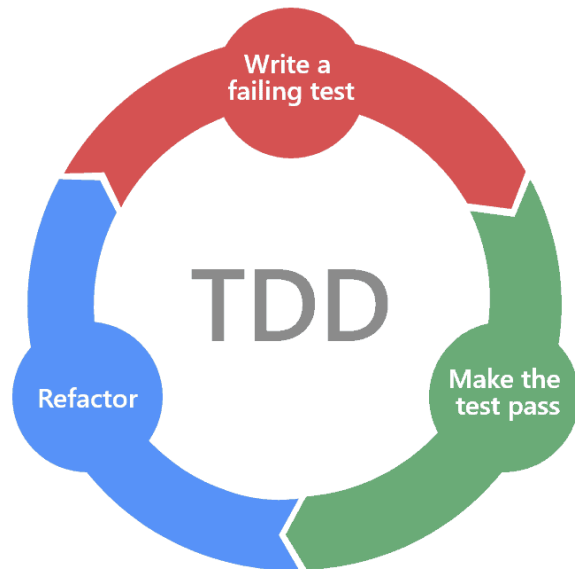
Test Driven Development (TDD)



TDD

Onderdeel van de Agile werkwijze

1. Schrijf een falende test
2. Zorg ervoor dat de test zo snel mogelijk groen ziet
3. Kuis je code op:
 - Duplicatie wegwerken
 - Methodes opsplitsen indien te groot



TDD voordelen

- Alle code getest + zeer hoge test coverage
- Meer focus op het gebruik van methodes omdat je steeds eerst over de test moet nadenken
- Minder fouten in de code
- Kleinere classes

TDD nadelen

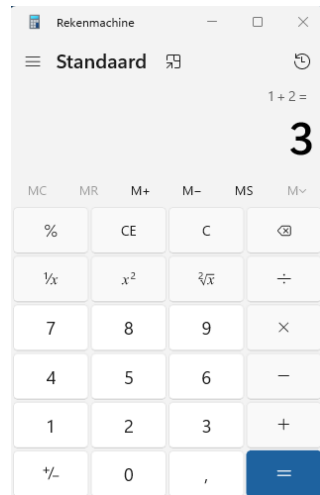
- Meer testcode schrijven (-> 3x productiecode)
- Meer tijd gespendeerd in testen
- Vals gevoel van veiligheid : veel testen != goed getest
- Hoge onderhoudskost → een kleine wijziging heeft mogelijk tot gevolg dat héél véél testen aangepast moeten worden

Voorbeeld

We maken een eenvoudige rekenmachine

Features:

- Twee gehele getallen optellen
- Bij het resultaat nog een geheel getal optellen



Voorbeeld - CalculatorTests

Test de Calculator class

1. addOneAndTwolsThree
2. addOneAndThreelsFour
3. addTwoAndThreelsFive
4. resultThreeAddOnelsFour
5. resultFourAddOnelsFive
6. resultFourAddTwolsSix
7. noResultAddOnelsOne

```
public class CalculatorTests {  
  
    @Test  
    void addOneAndTwoIsThree() {  
        Calculator calc = new Calculator();  
        assertEquals( expected: 3, calc.add( a: 1, b: 2));  
    }  
  
    @Test  
    void addOneAndThreeIsFour() {  
        Calculator calc = new Calculator();  
        assertEquals( expected: 4, calc.add( a: 1, b: 3));  
    }  
  
    @Test  
    void addTwoAndThreeIsFive() {  
        Calculator calc = new Calculator();  
        assertEquals( expected: 5, calc.add( a: 2, b: 3));  
    }  
  
    @Test  
    void resultThreeAddOneIsFour() {  
        Calculator calc = new Calculator();  
        calc.add( a: 1, b: 2);  
        assertEquals( expected: 4, calc.addToResult( a: 1));  
    }  
  
    @Test  
    void resultFourAddOneIsFive() {  
        Calculator calc = new Calculator();  
        calc.add( a: 1, b: 3);  
        assertEquals( expected: 5, calc.addToResult( a: 1));  
    }  
  
    @Test  
    void resultFourAddTwoIsSix() {  
        Calculator calc = new Calculator();  
        calc.add( a: 1, b: 3);  
        assertEquals( expected: 6, calc.addToResult( a: 2));  
    }  
}
```

TDD - nabeschuwing

1. Programmeren in kleine stapjes, niet te snel willen zijn.
2. Heel veel testen schrijven
 - creatief zijn
 - mogelijke fouten anticiperen

Productiecode : 11 lijnen

Testcode : 46 lijnen

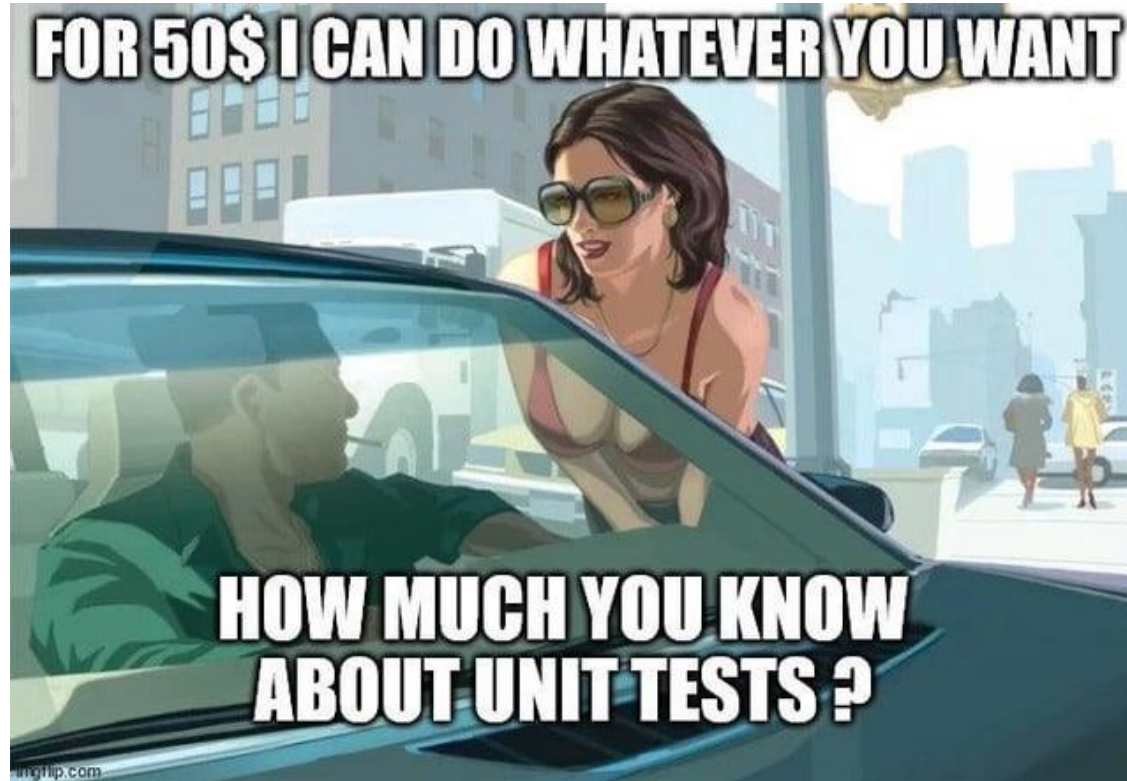
→ x4 !

Stel dat requirement wijzigt : gehele getallen -> decimale getallen

TDD – in de praktijk

- Company mind set
 - Veel bedrijven zeggen dat ze TDD werken maar in praktijk komt het er gewoon op neer dat ze ook testen schrijven
 - Enkel mogelijk indien iedereen (developers & management) er achter staat :
 - Langere doorlooptijd project
 - Hogere onderhoud kost
- Wel heel nuttig bij refactoring

Voor 2022 ...



Na 2022 ...



```
it('Clicks on the first link', () => {
  cy.get('a[href="/about/"]').click();
  cy.contains('About').should('be.visible');
});

it('Checks the header is present', () => {
  cy.get('header a').first().index(0).click();
  cy.wrap($el).click();
  cy.url().should('include', $el.attr('href'));
  cy.go('back');
});

it('fills out the contact form', () => {
  cy.get('a[href="/contact/"]').click();
  cy.get('input[name="name"]').type('John Doe');
  cy.get('input[name="email"]').type('john.doe@example.com');
});
```

ChatGPT in Testing

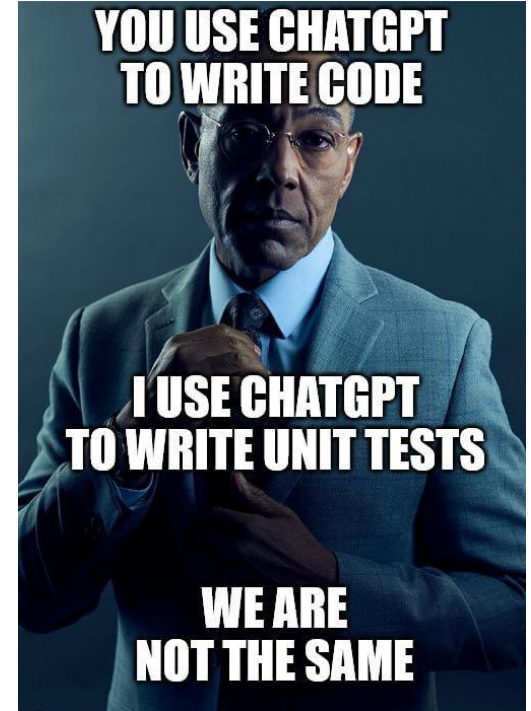
How ChatGPT Makes Testing Better

ChatGPT (of andere AI tools)

AI tools zijn er, gebruik ze waar ze goed voor zijn.

→ Repetitieve taken uitvoeren !

→ Unit testen schrijven !





**FOR TODAY
ANYWAY...**