Turn legacy code into clean code!

Jeroen Mengerink
David Baak

Agenda

- Introductie
- Legacy code voorbeeld
- Unittests maken
- Wat is refactoren
- Legacy code refactoren
- Wat zijn SOLID principes
- SOLID principes toepassen
- Conclusie

Introductie

Kom je wel eens niet werkende, geautomatiseerde testcases tegen waarvan de code zo complex en onleesbaar is dat reparatie erg moeilijk is?

In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

Lasagne code is code that has way too many layers. In object oriented programming, this means code that has lots of really small classes, while a few slightly larger classes would have been much more understandable.

Clean code

Clean code is simple and direct. Clean code reads like well-written prose.

Grady Booch

Programming is the art of telling another *human* what one wants the computer to do.

Donald Knuth

Clean code is a code that is written by someone who cares.

Michael Feathers

Leesbare code vereist goede naamgeving

- Namen moeten voor zich spreken
- Namen moeten geen overbodige informatie of implementatiedetails bevatten
- Namen moeten uit te spreken zijn
- Namen moeten relevant zijn in het probleemdomein

```
waitForIt()
findCustomerOrThrowExceptionIfNotFound(
        String s)
int qty_s_passed
```

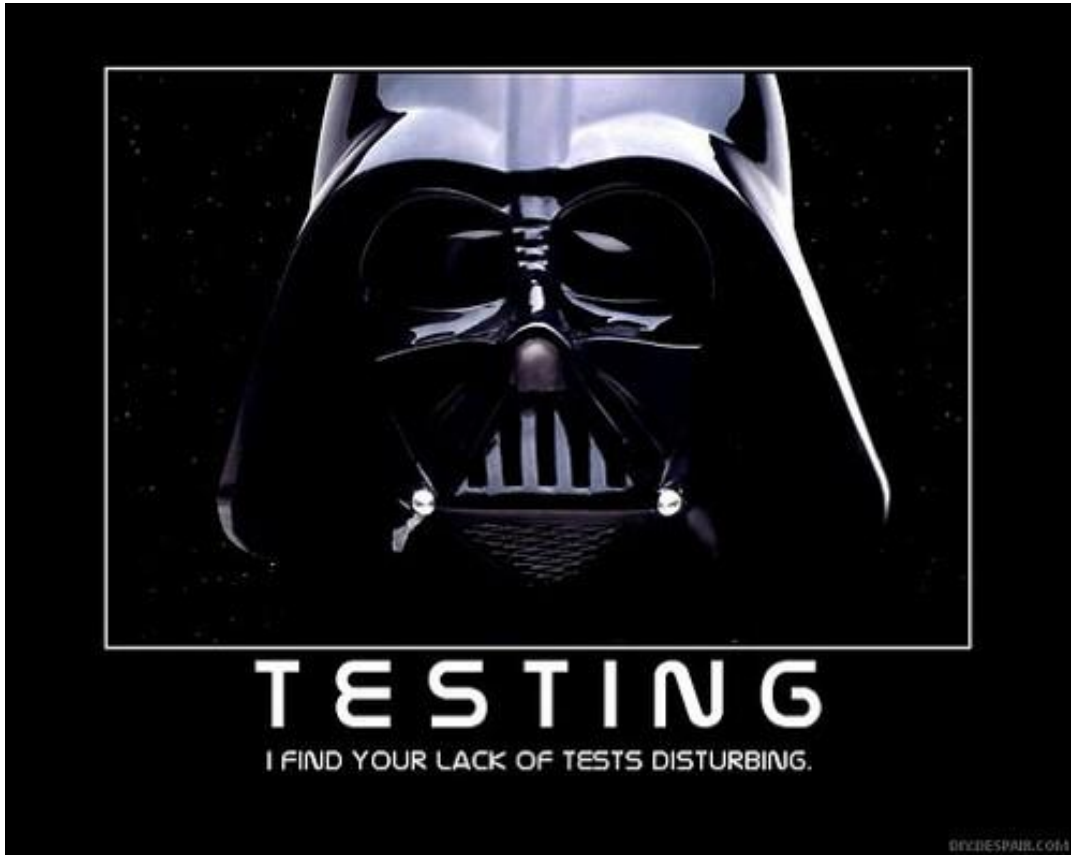
```
waitForAlert()
findCustomer(
        String customerName)
int passedScenarios
```

Legacy code voorbeeld

```
public class Encryptor {
  public String cryptWord(String word) {
    if (word.contains(" "))
      throw new InvalidParameterException();
    char[] wordArray = word.toCharArray();
    String newWord = "";
    for (int i = 0; i < word.length(); i++) {
      int charValue = wordArray[i];
      newWord += String.valueOf((char) (charValue + 2));
    }
    return newWord;
  }
```

De class is Encryptor dus moeten we iets kunnen encrypten

Laten we unittests maken



Refactoren

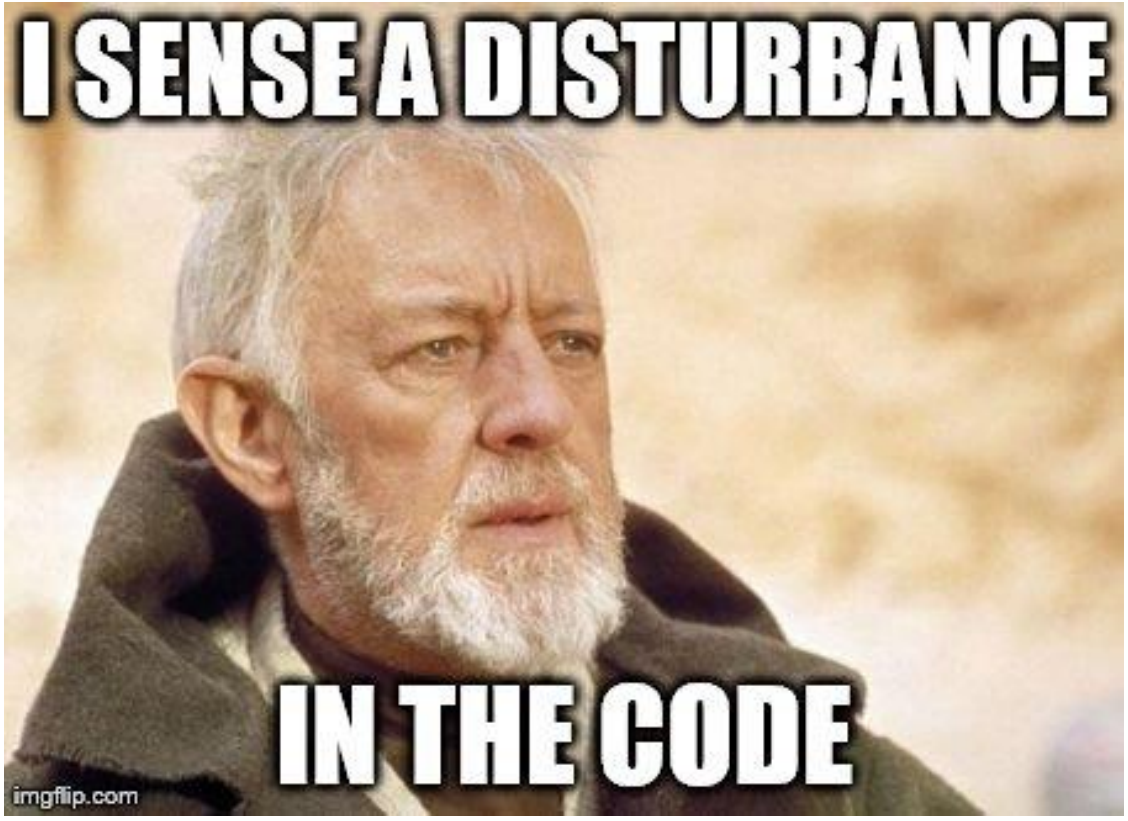
*"**Refactoren** is het **herstructureren** van de broncode van een computerprogramma met als doel de **leesbaarheid** en **onderhoudbaarheid** te verbeteren of het stuk code te **vereenvoudigen**. Het refactoren van broncode **verandert de werking van de software niet**: elke refactorstap is een kleine, ongedaan te maken stap die de leesbaarheid verhoogt zonder de werking aan te passen."*

(bron: https://nl.wikipedia.org/wiki/Refactoren)

Gebruik je IDE

- IDE’s ondersteunen refactoren, bijvoorbeeld:
  - Rename
  - Extract method
  - Change method signature
  - Inline
  - Extract constant
  - Convert local variable to field

Laten we refactoren



SOLID principes

- **S**ingle Responsibility
  - Elke software-entiteit (classes, modules, functies, etc.) moet de verantwoordelijkheid van één deel van de functionaliteit bevatten, niet meer of minder dan dat.

SOLID principes

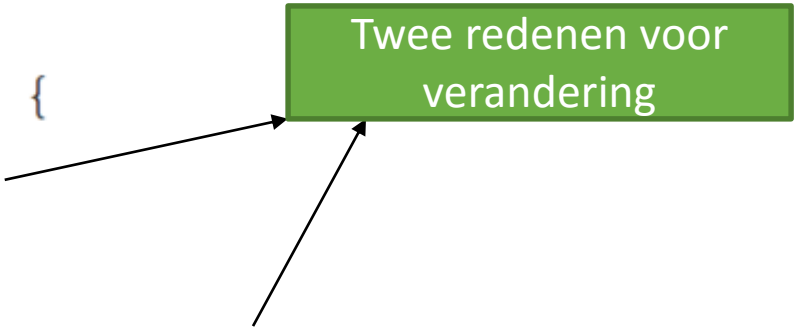
- **S**ingle Responsibility

```
public class UserManagement {

public void createUser() {}

public void sendConfirmationEmail() {}

}
```



Er mag maar één reden voor verandering zijn

SOLID principes

- **O**pen Closed
  - Software-entiteiten (classes, modules, functies, etc.) zouden open moeten zijn voor uitbreiding, maar gesloten voor verandering.

SOLID principes

- **S**ingle Responsibility
- **O**pen Closed

```
public void drawShape(Shape s) {
    if (s instanceof Circle) {
        drawCircle();
    } else if (s instanceof Triangle) {
        drawTriangle();
    }
}
```

Verandering als we Square introduceren

Code moet open zijn voor uitbreiding, maar gesloten voor verandering

SOLID principes

- **L**iskov Substitution
  - Als er gebruik wordt gemaakt van een base class, moet de refentie naar de base class kunnen worden vervangen door een afgeleide class zonder de werking te beïnvloeden.

SOLID principes

- **S**ingle Responsibility
- **O**pen Closed
- **L**iskov Substitution

```
public class Rectangle {
    protected int width;
    protected int height;
    public void setWidth(int width) {
        this.width = width;
    }
    public void setHeight(int height) {
        this.height = height;
    }
    public int getArea() {
        return width * height;
    }
}
```

```
public class Square extends Rectangle {
    public void setWidth(int width){
        this.width = width;
        height = width;
    }
    public void setHeight(int height){
        width = height;
        this.height = height;
    }
}
```

```
Rectangle r = new Square();
r.setHeight(5);
r.setWidth(10);
Invalid area!
```

Alle subtypes van een supertype moeten het supertype kunnen vervangen

SOLID principes

- **I**nterface Segregation
  - Deel grote interfaces op in kleinere, specifieke interfaces. Dan krijgen de classes die de interfaces implementeren alleen de methodes te zien die voor hen relevant zijn.

SOLID principes

- **S**ingle Responsibility
- **O**pen Closed
- **L**iskov Substitution
- **I**nterface Segregation

```
public interface Printer {
    void print();
    void scan();
    void email();
}
```

Niet elke printer kan emailen of scannen

Geen class moet afhankelijk zijn van methodes die hij niet gebruikt

SOLID principes

- **D**ependency Inversion
  - Methoden moeten afhankelijk zijn van abstracties, niet van concrete implementaties.

SOLID principes

- **S**ingle Responsibility
- **O**pen Closed
- **L**iskov Substitution
- **I**nterface Segregation
- **D**ependency Inversion

```
public class BusinessLoanValidator {
    public boolean isValid() {
        return true;
    }
}
```

```
public class Creditor {
    private final PersonalLoanValidator validator;
    public Creditor(PersonalLoanValidator validator) {
        this.validator = validator;
    }
    public void approveLoan() {
        if (validator.isValid()) {
            //process
        }
    }
}
```

Is nu afhankelijk van concrete implementatie

Alleen afhankelijkheid van abstractie, niet van concretie

Laten we SOLID toepassen

Conclusie

Behandel test automation als software development

- Verbeter de leesbaarheid en onderhoudbaarheid van legacy code in stappen:
  1. Maak unittests voor de legacy code
  2. Refactor de code
     - Gebruik juiste namen
     - Verwijder duplicatie
     - Pas SOLID principes toe

Vragen

David Baak – david.baak@polteq.com
Jeroen Mengerink – jeroen.mengerink@polteq.com