Test-Driven Design

Eigenschafter guter Unit Tests

Spezifisch: Fokus auf einen Aspekt des Codes

Prägnant: Wenig Code, sprechende Namen und Kommentare

Vollständig: Erwartetes korrektes *und* fehlerhaftes Verhalten

Schnell: Ermöglicht häufiges Testen – und auch viele Tests

Wiederholbar: Vermeidung von variierenden Faktoren
(Tageszeit, externe Server, Mondphase, ...)

Vorbehaltslos: Vermeidung von Annahmen über den Code

Test-Driven Development (TDD)

- Testen als Spezfikation
- Martin Fowler:

  "Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead."

$\Rightarrow$ Schreibe Tests *vor* dem Code.

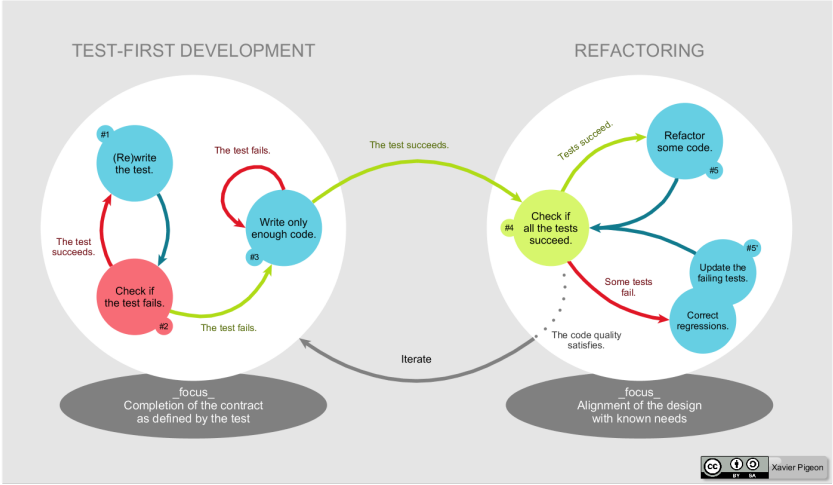
TDD Entwicklungszyklus

Fokus von Anfang an auf den Anforderungen. Diese treiben Design und Entwicklung voran.

- Starte mit einem Test
- Schreibe Code, so dass der Test erfolgreich durchläuft
- Schreibe gerade so viel Code, wie nötig
- Überarbeite den Code:

  Dopplungen, Umbenennungen, “magic numbers”
- Beginne von vorn

TDD development cycle



Behavior-Driven Design

Hintergrund

- Dan North (Coach für Agile Software Entwicklung)
- Häufige Frage in TDD-Kursen: "Whas soll ich testen?"
- Einige Erkenntnisse:
  - Beim TDD geht es eigentlich nicht ums Testen.
  - Fokus auf das, was eine Software Unit tun **sollte**
    $\Rightarrow$ **Verhalten** anstatt von **Test**
  - Anforderungen können als Verhalten ausgedrückt werden
  - Benenne den **Business Value**, um die nächste Aufgabe auszuwählen
- BDD bietet eine Sprache für die Anforderungsanalyse:
  - **Stories** & **scenarios**
  - "A story's behavior is simply its acceptance test"
  - 2003 Entwicklung von JBehave (als Ersatz für JUnit)

Feature: Stories & Scenarios

One Story

As a **ROLE**
I want the **FEATURE**
So that I have the **BENEFIT**

Multiple Scenarios

GIVEN some initial context
WHEN event occurs
THEN ensure some outcome

GIVEN some initial context
WHEN event occurs
AND different event occurs
THEN ensure other outcome

Gherkin

Menschenlesbare DSL um Features in BDD zu beschreiben (in JBehave & Cucumber)

```
Feature: Using the SimpleMath class

As a user of this class
I want to do basic arithmetic calculations
So I get correct results.

Scenario: Adding two positive numbers

Given a SimpleMath object
When I add 4 and 2
Then the result is 6

Scenario: Dividing by zero

Given a SimpleMath object
When I divide 1 by 0
Then an ArithmeticException is thrown
```

```
@When("I add (.*) and (.*)")
public void adding_two_numbers(List<String> animals) { ... }
```

Story

- User Story
  - Ein informaler Diskussionsstarter
  - Kein festes Format
    - To **receive benefit** as a **role**, I want **goal/desire**
    - As **who when where**, I **what** because **why**
    - ...
  - Es gibt Wege, um von "Use Cases" zu "User Stories" zu gelangen ($\Rightarrow$ UseCase 2.0)
- Einschränkungen
  - Wage und unvollständig
  - Nichtfunktionale Anforderungen sind schwierig
  - Übersichtlichkeit (Verwirrung und Überblick über das große Ganze)

Prinzipien

- Drei Grundwerte
  1. Wo ist der **Business Value**?
  2. Genug ist genug!
  3. Alles ist Verhalten!

- Ziel ist eine Spezfikation, die folgendes ist:
  - Ausführbar
  - Grundlage für eine “living documentation”
  - Definiert durch Verhaltensbeispiele
  - Entwickelt in einem iterativen, gemeinschaftlichen Prozess

Herangehensweise

Das Treffen der **3 Amigos**:

- Entwickler
- Tester
- Product Owner

- Gespräch über das *nächstwichtigste* Feature
- Extraktion eines Satzes *wichtiger* Szenarien
- Dokumentation (z. B. in Gherkin)
- Zurück zum Entwicklungszyklus

JBehave

Stories

A ISBN Format checker

Narrative:
In order to *have a uniform presentation of ISBN numbers*
As a *user*
I want to *have a formater for ISBN numbers*

Scenario: *An good ISBN number is formated correctly*
Given an ISBN format checker
When ISBN *123-4-567-89012-8 is* formated
Then ISBN equals 1234567890128

Scenario: *An bad ISBN number is formated correctly*

Given an ISBN format checker
When ISBN *123-4-567-89012-8-x is* formated
Then ISBN is empty

Scenario: *An empty ISBN number is formated correctly*

Given an ISBN format checker
When empty ISBN is formated
Then ISBN is empty

Scenario: *A already correctly formated ISBN is still formated correctly*

Given an ISBN format checker
When ISBN *1234567890128 is* formated
Then ISBN equals 1234567890128

Mapping functions

```java
package de.sopro;

import org.jbehave.core.annotations.Then;
import org.jbehave.core.annotations.When;

public class IsbnFormatCheckerSteps
{
    @When("empty ISBN is formated")
    public void whenEmptyISBNIsFormated()
    {
        // ...
    }


    @Then("ISBN is empty")
    public void thenISBNIsEmpty()
    {
        // ...
    }


    @Then("ISBN equals $formated_isbn")
    public void thenISBNEqualsFormatedIsbn(String formatedIsbn)
    {
        // ...
    }
}
```

Step parameters

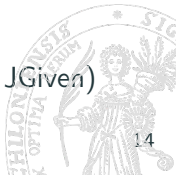
```
A ISBN Format checker

Narrative:
In order to have a uniform presentation of ISBN numbers
As a user
I want to have a formater for ISBN numbers


Scenario: A given ISBN is formated correctly

Given an ISBN format checker
When ISBN <isbm> is formated
Then ISBN equals <formated_isbm>


Examples:
|isbm|formated_ismb|
|1234567890128|1234567890128|
|123-4-567-89012-8|1234567890128|
|||
|123-4-567-89012-8-x||
```

Generelle Probleme

Es gibt viele Frameworks für viele Zielsprachen.

- Für Nichtprogrammierer kann es schwierig sein, Szenarien zu formulieren
  $\Rightarrow$ Letztendlich müssen Programmierer sie doch formulieren.
- Feature-Dateien und die Mapping-Funktionen müssen immer synchronisiert werden.
  $\Rightarrow$ Fehleranfällig
  $\Rightarrow$ Es gibt keinen "single source of truth"

- Es existieren Plugins für diverse IDEs
- Es gibt Ansätze keine DSL, sondern die Zielsprache nutzen zu nutzen ($\Rightarrow$ JGiven)



Abschluss

Eigenschaften guter Unit Tests

Spezifisch: Fokus auf einen Aspekt des Codes

Prägnant: Wenig Code, sprechende Namen und Kommentare

Vollständig: Erwartetes korrektes *und* fehlerhaftes Verhalten

Schnell: Ermöglicht häufiges Testen – und auch viele Tests

Wiederholbar: Vermeidung von variierenden Faktoren
(Tageszeit, externe Server, Mondphase, ...)

Verhaltensorientiert: Vermeidung von Annahmen über den Code

Quellen

- JBehave

  `https://jbehave.org`
- BDD Wiki

  `www.behaviourdriven.org`
- BDD Introduction/History (by Dan North)

  `https://dannorth.net/introducing-bdd`
- A Beginner's Guide (by Inviqa company)

  `https://inviqa.com/blog/bdd-guide`
- TDD development cycle

  `https://en.wikipedia.org/wiki/Test-driven_development`