Test Driven Development

Lukas Mösle

31.07.2019

Agenda

1. Test Driven Development
2. Tests
3. Vorgehen beim TDD
   - Beispiel
4. Testautomatisierung
5. Mein Vorgehen
6. Fazit

Thema

Test Driven Development eines RESTful Webservices mit dem Python Framework Django.

Test Driven Development

Test Driven Development

- Test First
  - Tests werden vor dem Code geschrieben

Test Driven Development

- Test First
  - Tests werden vor dem Code geschrieben
- Ziel
  - Clean Code that works

Test Driven Development

- Test First
  - Tests werden vor dem Code geschrieben
- Ziel
  - Clean Code that works
- Grundlegende Regeln
  - Neuer Code wird nur geschrieben, wenn ein automatisierter Test fehlschlägt
  - Duplikate eliminieren

Test Driven Development

- Test First
  - Tests werden vor dem Code geschrieben
- Ziel
  - Clean Code that works
- Grundlegende Regeln
  - Neuer Code wird nur geschrieben, wenn ein automatisierter Test fehlschlägt
  - Duplikate eliminieren
- Auswirkungen
  - Jeder Entwickler muss seine eigenen Tests schreiben
  - Schnelle Rückmeldung der Entwicklungsumgebung

Tests

Tests

- Software Testing soll die Qualität, Richtigkeit und Performance sicherstellen
- Unit und Functional Tests sind für das TDD relevant
- Weitere Tests sollten zusätzlich erstellt werden unabhängig von der Vorgehensweise

Tests

Unit Tests

- Testen einer einzelnen Unit
- Isoliert
- Kenntnis vom Code nötig

Functional/Integration Tests

- Fokus auf Anforderungen
- Verifiziert den Output

Tests

White Box Tests

- Code bekannt
- Interne Strukturen testen
- Isoliert
- Unit Tests

Grey Box Tests

- Beim TDD werden White Box Tests erstellt vor dem Code

Black Box Tests

- Code nicht bekannt
- Testen der Anforderungen
- Funktions-orientiert
- Functional Tests

Isolierte Tests

- Tests sollten unabhängig sein
- Tests sollten schnell
- Jeder fehlschlagende Test ein Fehler
- Unabhängig von Ausführungsreihenfolge

Vorgehen beim TDD

Vorgehen - Red Green Refactor

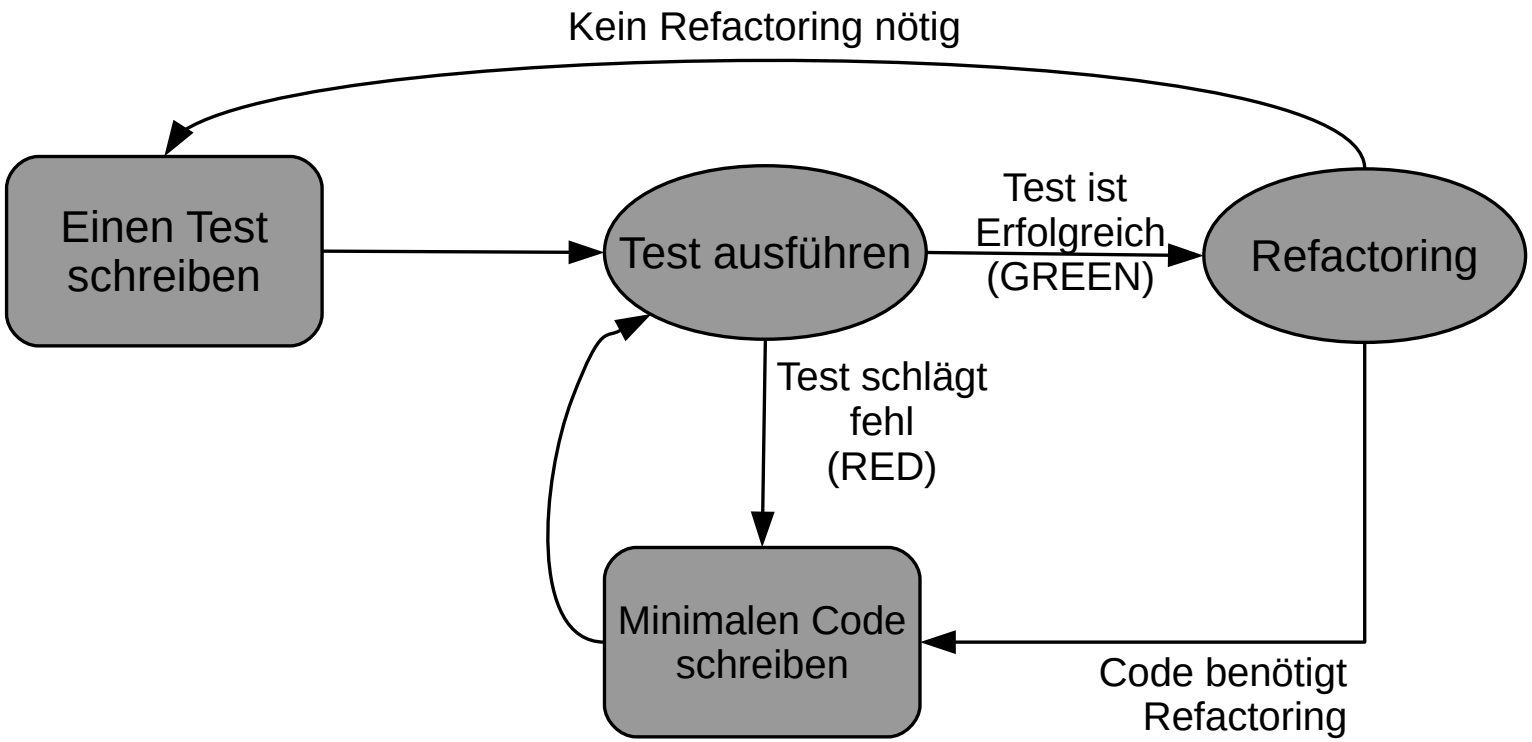


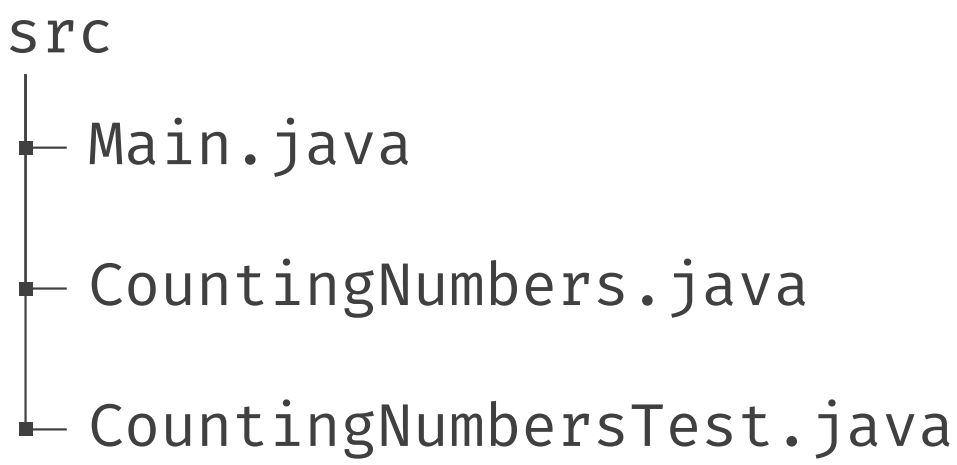
Abbildung: Red Green Refactor

Vorgehen beim TDD

Beispiel

Ein kleines Beispiel

Eine Funktion, die die Anzahl der Buchstaben in einem String zählt.



▸ Code

Ein kleines Beispiel

Zuerst einen Test schreiben, der überprüft ob eine Zahl zurückgegeben wird.

CountingNumbersTest.java

```
class CountingNumbersTest {
    CountingNumbers countingNumbers;

    @BeforeEach
    public void setUp() {
        this.countingNumbers = new CountingNumbers();
    }

    @Test
    public void testCountingNumbersReturnsInteger() {
        assertEquals(this.countingNumbers.countNumbers("Test").
            getClass(), Integer.class);
    }
}
```

Ein kleines Beispiel

Den Test erfüllen, indem eine einfache Methode geschrieben wird, die eine Zahl zurück liefert.

CountingNumbers.java

```
public Integer countNumbers() {
    return 42;
}
```

Ein kleines Beispiel

Nun einen zweiten Test erstellen, der der Methode einen String übergibt und als Ergebnis die Anzahl der Buchstaben erwartet.

CountingNumbersTest.java

```
@Test
public void testCountingNumbersReturnsTheNumberOfCharacters() {
    assertEquals(this.countingNumbers.countNumbers("Hello World"),
          11);
}
```

Ein kleines Beispiel

Jetzt wird die Logik der Funktion implementiert.

CountingNumbers.java

```
public Integer countNumbers(String textToCountCharacters) {
    return textToCountCharacters.length();
}
```

Ein kleines Beispiel

Jetzt wird die Logik der Funktion implementiert.

CountingNumbers.java

```
public Integer countNumbers(String textToCountCharacters) {
    return textToCountCharacters.length();
}
```

Der erste Test schlägt fehl

Der Test wird angepasst, indem der Methode ein String übergeben wird.

```
@Test
public void testCountingNumbersReturnsInteger() {
    assertEquals(this.countingNumbers.countNumbers("Test").
        getClass(), Integer.class);
}
```

Vorgehen - Double Loop TDD

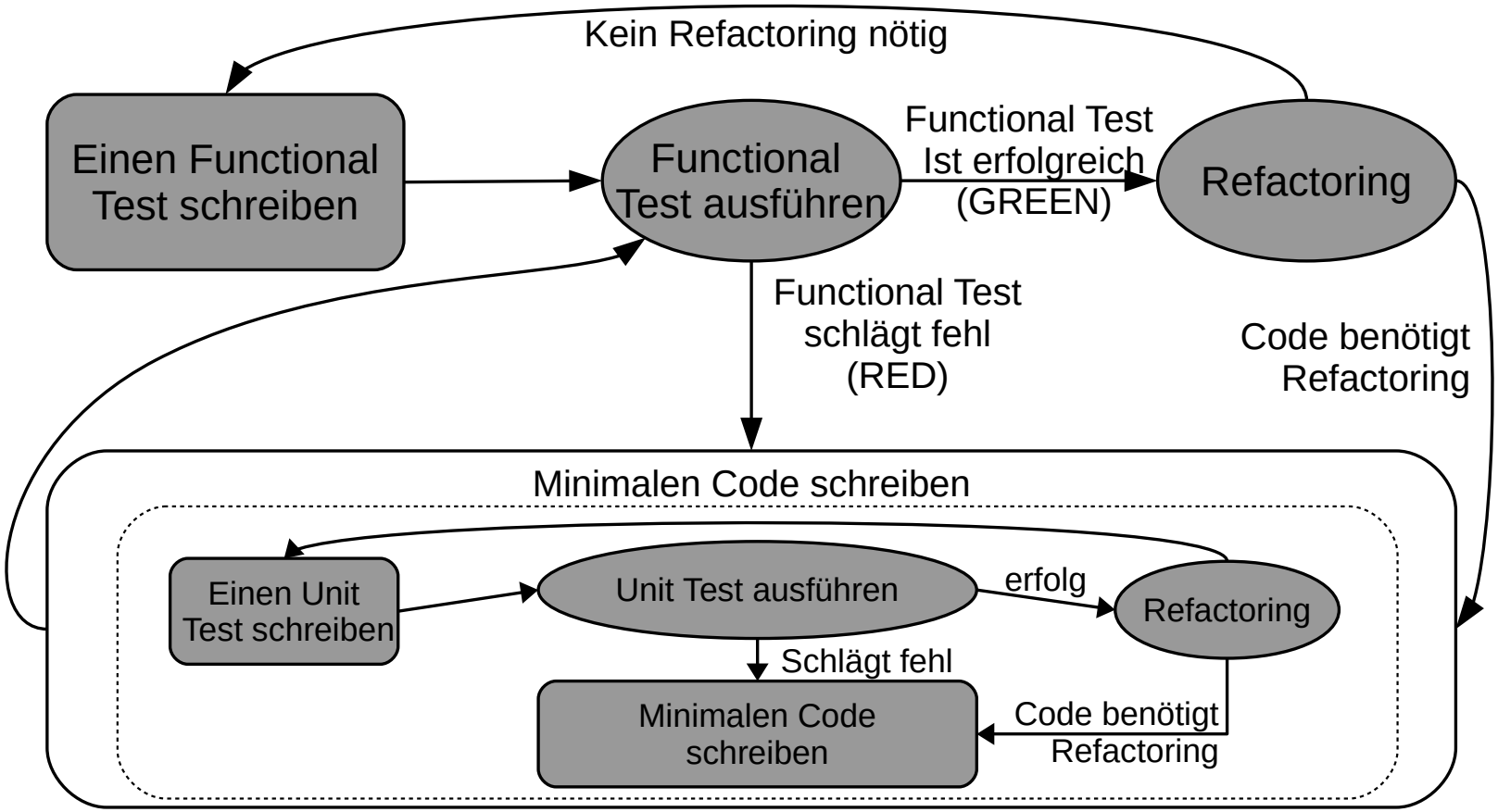


Abbildung: Double Loop TDD

Testautomatisierung

Testautomatisierung

- Software automatisiert testen und deployen
- Continuous Integration, Delivery, Deployment

Testautomatisierung

- Software automatisiert testen und deployen
- Continuous Integration, Delivery, Deployment
- **Ziel**
  - Fehler beim build und deployment minimieren

Abbildung: Übersicht Continuous Integration, Delivery und Deployment

Mein Vorgehen

Mein Vorgehen

1. Recherche und TDD ausprobieren
2. Technisches Setup
3. Erste Funktionen implementiert
4. CI - CD Pipeline erstellt
5. Implementierung
6. Bachelorarbeit geschrieben

Mein Vorgehen bei der Entwicklung

Story

- Was macht die Funktion?
- Was sind mögliche Test Szenarien?

Mein Vorgehen bei der Entwicklung

Story

- Was macht die Funktion?
- Was sind mögliche Test Szenarien?

Entwicklung

- Double Loop TDD
- Story zu Functional Test

Mein Vorgehen bei der Entwicklung

Story	■ Was macht die Funktion? ■ Was sind mögliche Test Szenarien?
Entwicklung	■ Double Loop TDD ■ Story zu Functional Test
Dokumentation	■ Open API Dokumentation des entstandenen Endpoints

Fazit

Vor- und Nachteile

Vorteile

- Sauberer Code
- Gute Wartbarkeit
- Hohe Testabdeckung
- Kein/Kaum redundanten Code
- Klares Vorgehen

Nachteile

- Höhere Zeitaufwand

Herausforderungen

- Disziplin erforderlich
- Gute Tests schreiben ist anspruchsvoll

Vor- und Nachteile

Vorteile

- Sauberer Code
- Gute Wartbarkeit
- Hohe Testabdeckung
- Kein/Kaum redundanten Code
- Klares Vorgehen

Nachteile

- Höhere Zeitaufwand

Herausforderungen

- Disziplin erforderlich
- Gute Tests schreiben ist anspruchsvoll

Fazit

Wenn TDD falsch gemacht wird ergeben sich keine/kaum Vorteile.

Empfehlungen

- Double Loop TDD in Kombination mit Selenium sinnvoll
  - Double Loop TDD weniger sinnvoll bei der API Entwicklung, da Unit und Funktional Tests nahezu identisch sind
- TDD ist sehr praktisch bei der API Entwicklung
- **YAGNI** - You Aren't Gonna Need It
  - Am besten nicht mit dem Projektsetup und der Konfiguration beginnen
  - Hilfsklassen erst implementieren wenn diese gebraucht werden
- **KISS** - Keep it simple, stupid
- Fehlschlagende Tests nicht auskommentieren/löschen
- Nicht abschrecken lassen vom Mehraufwand - TDD zeigt seine Vorteile beim Refactoring.

Habt ihr Fragen?