Schreib‘ die Tests, die du auch lesen magst!

andrena objects ag

Claudia Fuhrmann

André Kappes

23. Juli 2020
SAEC Days / Clean Code Days

Habt Ihr…

… schon einmal an einem roten Test gesessen und nicht verstanden, was er eigentlich bewirken soll?

… diesen Test selbst geschrieben?

Code-Typografie

Namensgebung

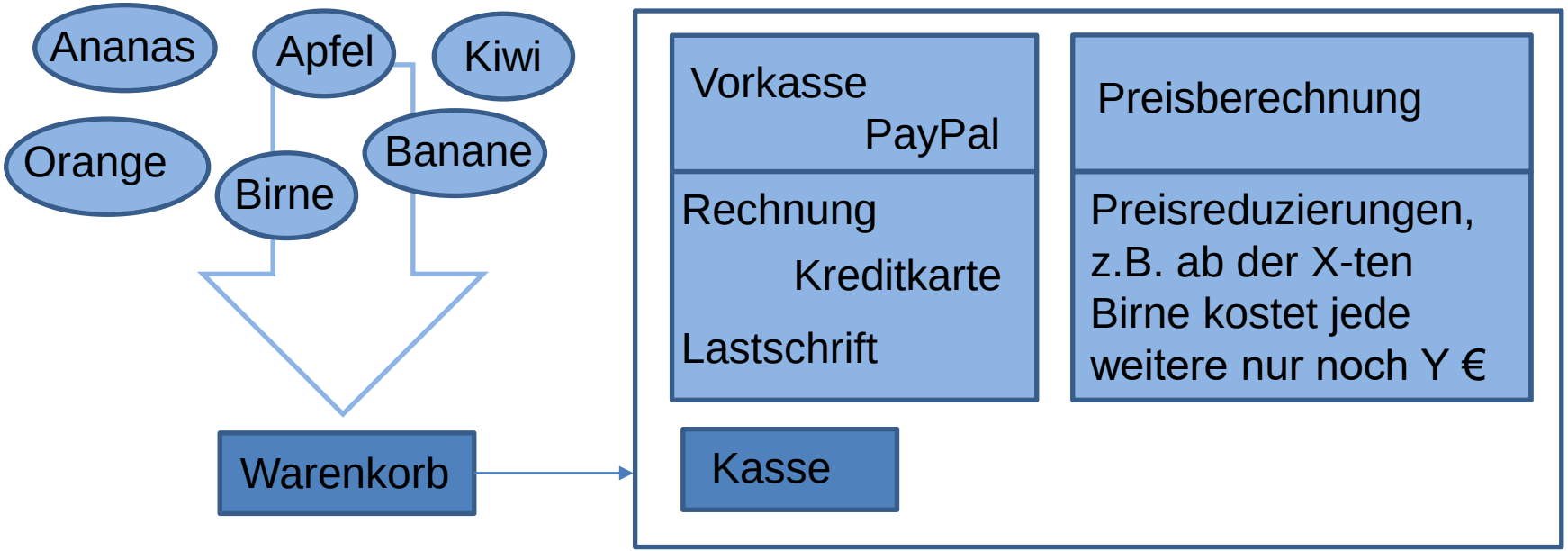
Lesbare Tests

Übersichtlichkeit

Prägnanz

Das Projekt

Online - Obstversand



Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Regeln aus der Typografie

- Lange Zeilen bzw. Umbrüche vermeiden
- Durch Absätze strukturieren
- AAA-Pattern zur Orientierung:
  Gliederung des Tests in Abschnitten nach
  Arrange – Act – Assert

Strukturierung mit dem AAA-Pattern

```
@Test
void testAdd() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(ObstTyp.BIRNE,
           PreisStrategyFactory.createNormalPreisStrategy(Money.of(0.70)));
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(3, ObstTyp.BIRNE);
    List<Posten> posten = warenkorb.getPosten();
    assertEquals(ObstTyp.BIRNE, posten.get(0).getTyp());
    assertEquals(8, posten.get(0).getAnzahl());
    assertEquals(Money.of(5.60), posten.get(0).getPreis());
}
```

Strukturierung mit dem AAA-Pattern

```
@Test
void testAdd() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(ObstTyp.BIRNE,
            PreisStrategyFactory.createNormalPreisStrategy(Money.of(0.70)));
    Warenkorb warenkorb = new Warenkorb(preisRepository);

    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(3, ObstTyp.BIRNE);
    List<Posten> posten = warenkorb.getPosten();

    assertEquals(ObstTyp.BIRNE, posten.get(0).getTyp());
    assertEquals(8, posten.get(0).getAnzahl());
    assertEquals(Money.of(5.60), posten.get(0).getPreis());
}
```

Arrange

Act

Assert

Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Wie soll sie*denn heißen?

Sprechende Namen finden

*) Variable, Testmethode

Sprechende Namen finden

```
class BestellServiceTest {
    private KundenRepository repositoryMock = mock(KundenRepository.class);
    private BestellService service = new BestellService(repositoryMock);

    @Test
    void berechnePreis() {
        PreisRepository repository = new PreisRepository();
        repository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));

        Warenkorb input = new Warenkorb(repository);
        input.add(5, BIRNE);

        BestellUebersicht bU = service.getBestellUebersicht(input);

        assertEquals(Money.of(3.30), bU.getGesamtpreis());
    }
}
```

Gute Namen

- Zweck der Variablen / des Felds in diesem Testfall erklären
- Generische Namen vermeiden – spezifische Namen bevorzugen
- Lesbar / aussprechbar
- Je kleiner der Scope, desto kürzer der Name
- Unterscheidbare Namen verwenden

Magic Values

```
@Test
void berechnePreis() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));

    Warenkorb mit5Birnen = new Warenkorb(preisRepository);
    mit5Birnen.add(5, BIRNE);

    BestellUebersicht uebersicht = bestellService.getBestellUebersicht(mit5Birnen);

    assertEquals(Money.of(3.30), uebersicht.getGesamtpreis());

}
```

Warum 3.30?

Magic Values

```
@Test
void berechnePreis() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(BIRNE, reduzierterPreisAb(5, of(0.70), of(0.50)));

    Warenkorb mit5Birnen = new Warenkorb(preisRepository);
    mit5Birnen.add(5, BIRNE);

    BestellUebersicht uebersicht = bestellService.getBestellUebersicht(mit5Birnen);

    assertEquals(PREIS_FUER_VIER_BIRNEN.plus(REDUZIERTER_PREIS_FUER_FUENFTE_BIRNE),
            uebersicht.getGesamtpreis());
}
```

Erklärende Konstanten

Testmethoden benennen

Namenskonvention von Roy Osherove:

Function/Feature – Input/State – Outcome/Behaviour

```
@Test
void berechnePreis () {
    ...
}
```

```
@Test
void berechnePreis_reduktionAb5Birnen_wirdBeachtet() {
    ...
}
```

Empfehlungen

- Erklärende Namen für lokale Variablen und Felder wählen
- Testmethode aussagekräftig benennen
- Magic Values vermeiden

Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Prägnanz

```
private Posten EINE_ANANAS = Posten.of(1, ANANAS, Money.of(3.00));
private Posten EINE_BANANE = Posten.of(1, BANANE, Money.of(2.00));
private Posten EIN_APFEL = Posten.of(1, APFEL, Money.of(1.50));
private Posten EINE_BIRNE = Posten.of(1, BIRNE, Money.of(1.00));
private Posten EINE_KIWI = Posten.of(1, KIWI, Money.of(3.50));
private Posten EINE_ORANGE = Posten.of(1, ORANGE, Money.of(2.50));

private PreisRepository preis
private KundenRepository kur

@BeforeEach
void setUp() {
    preisRepository = new PreisRepository();
    PreisRepositoryInitializer.create(preisRepository).init();
    kundenRepository = new KundenRepository();
}
```

```
@Test
void testWarenkorb() {
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    warenkorb.add(1, ANANAS);
    warenkorb.add(1, BANANE);
    warenkorb.add(1, APFEL);
    warenkorb.add(1, BIRNE);
    warenkorb.add(1, KIWI);

                                    denId();
                                    ;
    List<Posten> posten = warenkorb.getPosten();

    assertThat(posten).contains(EINE_ANANAS);
    assertThat(posten).contains(EINE_BANANE);
    assertThat(posten).contains(EIN_APFEL);
    assertThat(posten).contains(EINE_BIRNE);
    assertThat(posten).contains(EINE_KIWI);
    assertThat(posten).contains(EINE_ORANGE);
}
```

Zu lang und unübersichtlich!

KundenId setzen

Posten im Warenkorb

Fragen, die ich mir stellen sollte, wenn der Test zu lang ist:

- Kann man den Test aufteilen, weil nicht alles in diesen Test gehört?
- Gehören die einzelnen Asserts in eine andere Testklasse?
- Hat die getestete Klasse zu viele Verantwortlichkeiten?

Prägnanz

```
@Test
void kundenIdIstNichtNull() {
    Warenkorb warenkorb = new Warenkorb(preisRepository);
    UUID kundenId = warenkorb.getKundenId();
    assertThat(kundenId).isNotNull();
}
```

Kurze Tests

```
@Test
void getPosten() {
    Warenkorb warenkorb = WarenkorbBuilder.get()
                                          .addAnanas(1)
                                          .addBananen(1).create();

    List<Posten> posten = warenkorb.getPosten();
    assertThat(posten).containsExactlyInAnyOrder(EINE_ANANAS, EINE_BANANE);
}
```

Wenige Zusicherungen

Empfehlungen

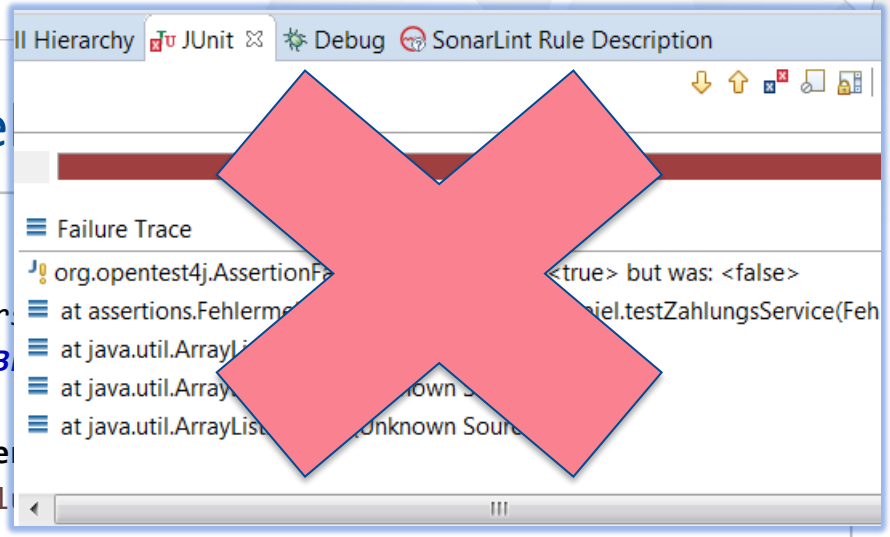
Tests sollten:

- kurz sein
- wenige Zusicherungen haben, am besten nur eine
- in den dazugehörigen Testklassen sein

Wenn dein Test nicht mehr auf deinen Bildschirm passt, ist er definitiv zu lang!

Sprechende Zusicherungen und Fe

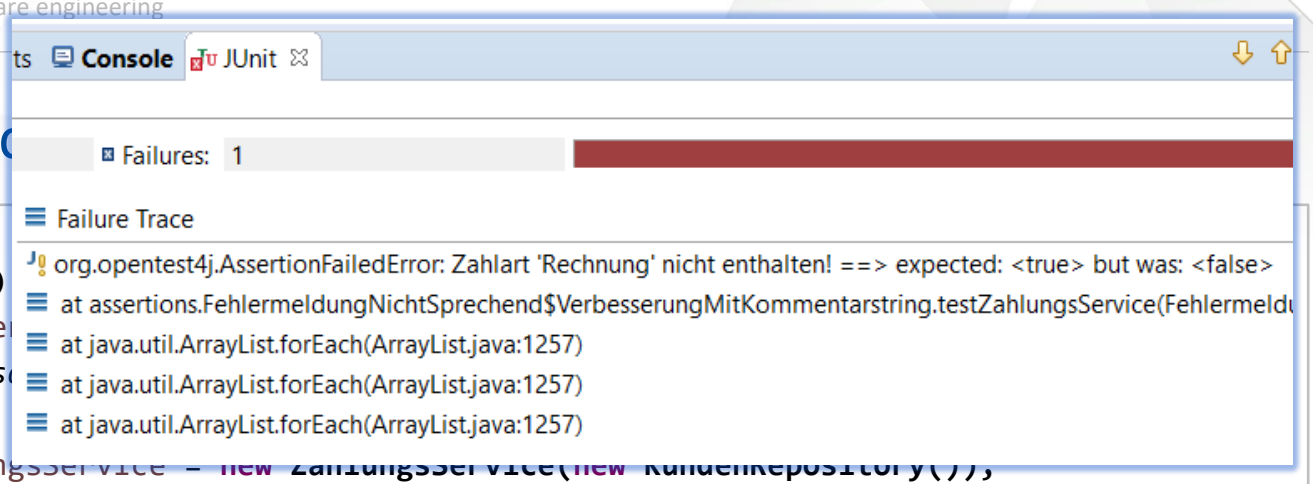
```
@Test
void testZahlungsService() {
    BestellUebersicht ueberLimit = mock(BestellUeber
    when(ueberLimit.getGesamtpreis()).thenReturn(UEB

    ZahlungsService zahlungsService = new ZahlungsSe
    List<ZahlungsArt> angeboteneZahlungsArten = zahl

    assertTrue(!angeboteneZahlungsArten.contains(ZahlungsArt.RECHNUNG));
}
```



Keine Informationen über den Grund eines Fehlschlags

Negation

Sprechende Zusic



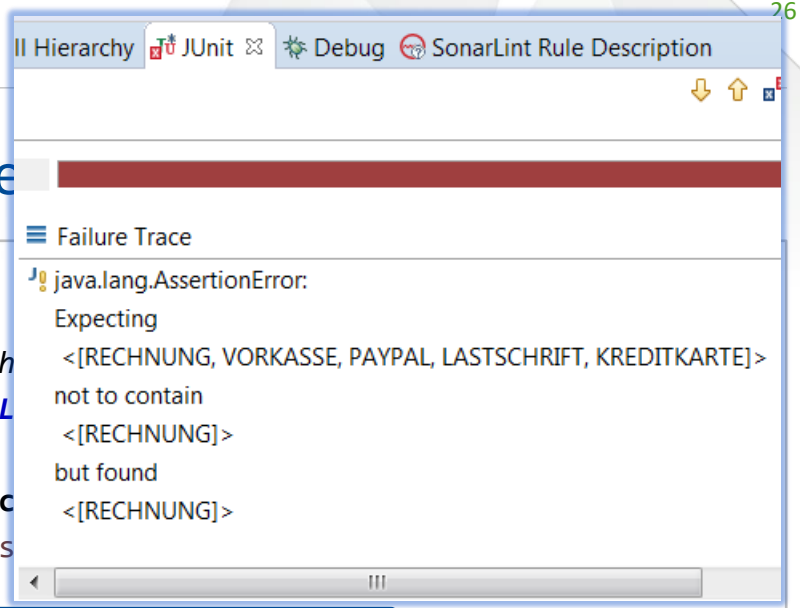

```
@Test
void testZahlungsService()
    BestellUebersicht ueber
    when(ueberLimit.getGes

    ZahlungsService zahlungsService = new ZahlungsService(new KundenRepository());
    List<ZahlungsArt> angeboteneZahlungsArten = zahlungsService.getZahlungsarten(ueberLimit);

    assertTrue(!angeboteneZahlungsArten.contains(ZahlungsArt.RECHNUNG),"Zahlart 'Rechnung' nicht
                                                                        enthalten!");
}
```

Aussagekräftiger Kommentarstring

Sprechende Zusicherungen und Fehle

```
@Test
void testZahlungsService() {
    BestellUebersicht ueberLimit = mock(BestellUebersich
    when(ueberLimit.getGesamtpreis()).thenReturn(UEBER_L

    ZahlungsService zahlungsService = new ZahlungsServic
    List<ZahlungsArt> angeboteneZahlungsArten = zahlungs

    assertThat(angeboteneZahlungsArten).doesNotContain(ZahlungsArt.RECHNUNG);
}
```

Assertj Matcher



Empfehlungen

- Zusicherungen möglichst präzise formulieren
- Auf sprechende Fehlermeldung im Testfehlschlags-Fall achten

Code-Typografie

Namensgebung

Lesbare Tests

Übersichtlichkeit

Prägnanz

Hin- und Herspringen

Warum? Was gibt es dort?

```
public class BestellUebersichtTest extends WarenkorbTest {

    private Posten EINE_BANANE = Posten.of(1, BANANE, Money.of(2.00));
    private Posten EINE_ANANAS = Posten.of(1, ANANAS, Money.of(3.00));

    private PreisRepository preisRepository = new PreisRepository();
    private KundenRepository kundenRepository;
    private BestellService underTest;

    @BeforeEach
    void setUp() {
        PreisRepositoryInitializer.create(preisRepository).init();
        underTest = new BestellService(kundenRepository);
    }
```

Wie sehen die Preise aus?

Hin- und Herspringen

```
@Test
void getBestellUebersicht_eineAnanasUndEineBanane_Bestell...sichtOk()
    Warenkorb warenkorb = createWarenkorbWithAnanasUndBanane(preisRepository);

    BestellUebersicht bestellUebersicht = underTest.getBestellUebersicht(warenkorb);

    assertBestellUebersicht(bestellUebersicht, warenkorb.getKundenId(), Money.of(5.00),
                                                                    EINE_ANANAS, EINE_BANANE);
}

private void assertBestell...sicht(BestellUebersicht bestellUebersicht, UUID kundenId,
                                                                    Money preis, Posten... posten) {
    assertThat(bestellUeb...reis()).isEqualTo(preis);
    assertThat(bestellUeb...sten()).containsExactlyInAnyOrder(posten);
    assertThat(bestellUebersicht.getKundenId()).isEqualTo(kundenId);
}
```

Wo finde ich diese Methode?

Was testet dieses Assert?

Hin- und Herspringen

```
public class BestellServiceTest {
    private Money PREIS_EINE_ANANAS = Money.of(3.00);
    private Money PREIS_EINE_BANANE = Money.of(2.00);
    private Posten EINE_ANANAS = Posten.of(1, ANANAS, PREIS_EINE_ANANAS);
    private Posten EINE_BANANE = Posten.of(1, BANANE, PREIS_EINE_BANANE);

    private KundenRepository kundenRepository;
    private BestellService underTest;

    @BeforeEach
    void setUp() {
        kundenRepository = new KundenRepository();
        underTest = new BestellService(kundenRepository);
    }
```

Keine magische Preiserstellung

Hin- und Herspringen

```
@Test
void getBestellUebersicht_eineAnanasUndEineBana… bestellUebersich…
    Warenkorb warenkorb = WarenkorbBuilder.warenkorb().withKundenId(KUNDEN_ID)
                                                      .addAnanas(1, PREIS_EINE_ANANAS)
                                                      .addBananen(1, PREIS_EINE_BANANE).create();

    BestellUebersicht bestellUebersicht = underTest.getBestellUebersicht(warenkorb);

    assertThat(bestellUebersicht)
            .hasGesamtpreis(PREIS_EINE_ANANAS.plus(PREIS_EINE_BANANE))
            .hasPosten(EINE_ANANAS, EINE_BANANE)
            .hasKundenId(KUNDEN_ID);
}
```

Builder, der den Warenkorb samt Preise erstellt

Custom matcher

Empfehlungen und Lösungsansätze

- Relevantes sichtbar machen, Irrelevantes verbergen
- Builder Pattern
- Custom Matchers

Noise stört die Lesbarkeit

```
@Test
void testAdd() {
    PreisRepository preisRepository = new PreisRepository();
    preisRepository.save(ObstTyp.BIRNE,
            PreisStrategyFactory.createNormalPreisStrategy(Money.of(0.70)));
    Warenkorb warenkorb = new Warenkorb(preisRepository);

    warenkorb.add(5, ObstTyp.BIRNE);
    warenkorb.add(3, ObstTyp.BIRNE);
    List<Posten> posten = warenkorb.getPosten();

    assertEquals(ObstTyp.BIRNE, posten.get(0).getTyp());
    assertEquals(8, posten.get(0).getAnzahl());
    assertEquals(Money.of(5.60), posten.get(0).getPreis());
}
```

Weniger Noise durch Syntactic Sugar

Static imports verwenden

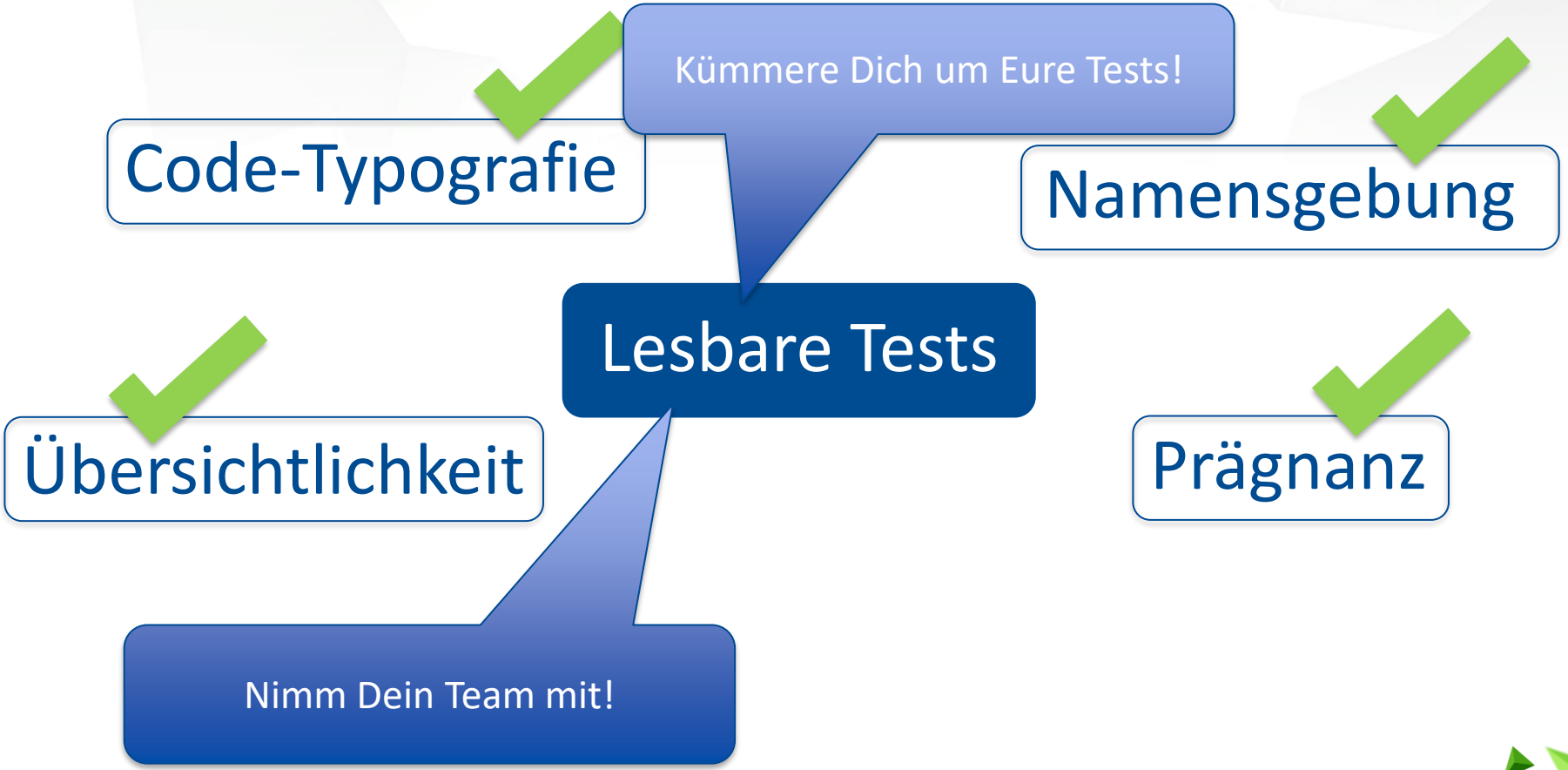
```
@Test
void testAdd() {
    PreisRepository repository = new PreisRepository();
    repository.save(BIRNE, normalPreis(0.70));
    Warenkorb warenkorb = new Warenkorb(repository);

    warenkorb.add(5, BIRNE);
    warenkorb.add(3, BIRNE);

    Posten soleItem = warenkorb.getPosten().get(0);

    assertEquals(BIRNE, soleItem.getTyp());
    assertEquals(8, soleItem.getAnzahl());
    assertEquals(Money.of(5.60), soleItem.getPreis());
}
```

Syntactic sugar-Methoden extrahieren

```
private PreisStrategy normalPreis(double value) {
    return PreisStrategyFactory.createNormalPreisStrategy(of(value));
}
```

Erklärende Variablen einführen

Zusammenfassung



Referenzen

Bücher

- Roy Osherove, **The Art of Unit Testing** (Manning, 2013)
- Gerard Meszaros, **xUnit Test Patterns: Refactoring Test Code** (Addison Wesley, 2007)
  http://xunitpatterns.com/

Blogs

- Petri Kainulainen, **Writing Clean Tests**
  https://www.petrikainulainen.net/writing-clean-tests/
- Thomas Countz, **Essential & Relevant: A Unit Test Balancing Act**
  https://8thlight.com/blog/thomas-countz/2019/02/19/essential-and-relevant-unit-tests.html

Vielen Dank!

Feedback

- Claudia Fuhrmann
  claudia.fuhrmann@andrena.de

- André Kappes
  andre.kappes@andrena.de