codecentric
agile software factory

Clean Test Code

Clean Code Days 12.11.2014

David Völkel, @davidvoelkel

@davidvoelkel

Entwickler & Consultant

codecentric

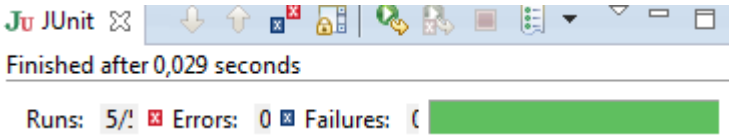
Software Craftsman

- Test-Driven Development
- Softwaredesign
- @softwerkskammer

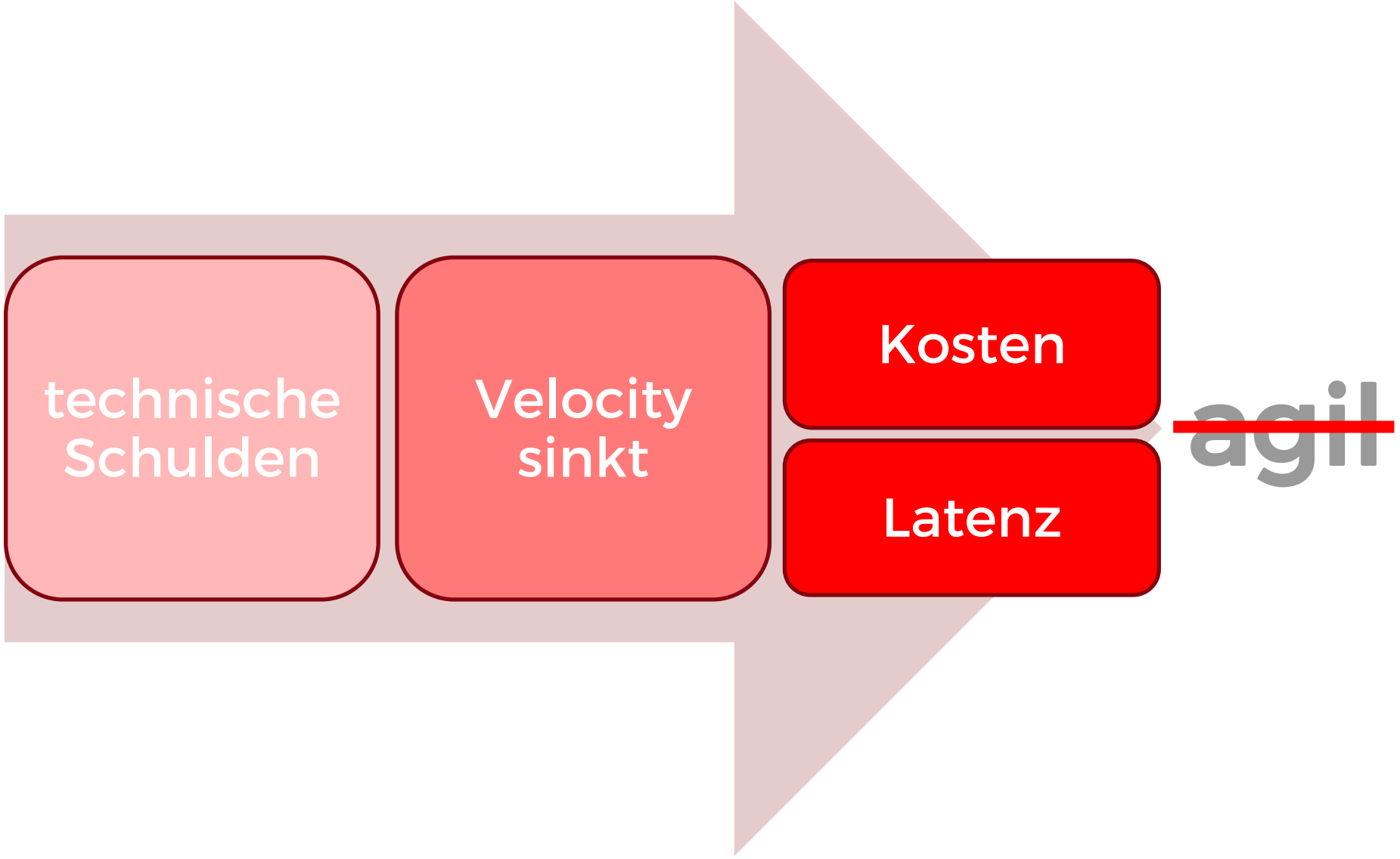
Agil

Automatisierte Tests

=> häufige Releases + Qualität



Dirty Test Code



codecentric
agile software factory

Clean Tests

4 Eigenschaften
- verlässlich
- lesbar
- redundanzfrei
- fokussiert

Fazit

Fragen & Diskussion

Testphasen

	xUnit-Patterns		BDD	Funktionale Tests
Input	setup	arrange	given	Eingabe
	execute	act	when	“Fixture“
Output	verify	assert	then	Ausgabe
	[teardown]	[annihilate]		

4 Rules of Simple Design

1. Pass all Tests
2. Clear, Expressive & Consistent
3. No Duplication
4. Minimal Units

Verlässlich

Gefahr Komplexität, vermeide

- boolsche Assertions

```
assertTrue(age >= 18);
```

Verlässlich

Gefahr Komplexität, vermeide

- boolsche Assertions

```
assertTrue(age >= 18);
assertThat(age, greaterThan(17))
```

Verlässlich

Gefahr Komplexität vermeide

- boolsche Assertions
- Conditionals, komplexe Asserts & Schleifen

```
boolean containsName = false;
for (String name: names) {
    if (name.equals("David")) {
        containsName = true;
    }
}
assertTrue(containsName);
```

Verlässlich

Gefahr Komplexität vermeide

- boolsche Assertions
- Conditionals, komplexe Asserts & Schleifen

```
boolean containsName = false;
for (String name: names) {
    if (name.equals("David")) {
        containsName = true;
    }
}
assertTrue(containsName);
```

```
assertThat(names, hasItem("David"));
```

Lesbar

- *„Write the tests you want to read“*
- 80% lesen, 20% schreiben
- Executable Specifications
- Living Documentation

Abstraktion

Weglassen von Details

Testmethode

Gerard Meszaros:

„When it is not important for something to be seen in the test method, it is important that it not be seen in the test method! “

„Im Bestellformular soll der Benutzer mit dem Abbrechen-Button den Bestellprozess beenden können”

„Im Bestellformular soll der Benutzer mit dem Abbrechen-Button den Bestellprozess beenden können"

```
@Test public void prozessAbbruch_inBestellFormular() {
    server.laufendeProzesse(0);
    ProzessStatus prozess = server.starteProzess(Prozess.BESTELLUNG);
    prozessInstanzId = prozess.getInstanzId();

    server.laufendeProzesse(1);
    server.erwarteBenutzerFormular(prozessInstanzId, BESTELL_FORMULAR);

    Formular formular = new Formular();
    formular.setBestellDatum("01.01.2015");
    formular.setStandardVersand(false);
    String taskId = server.taskIdZu(prozessInstanzId);
    controller.submitEingabe(formular,
         Mockito.mock(BindingResult.class), null,
         taskId, null, ABBRUCH);

    server.laufendeProzesse(0);
}
```

„Im Bestellformular soll der Benutzer mit dem Abbrechen-Button den Bestellprozess beenden können"

```
@Test public void prozessAbbruch_inBestellFormular() {
    server.laufendeProzesse(0);
    ProzessStatus prozess = server.starteProzess(Prozess.BESTELLUNG);
    prozessInstanzId = prozess.getInstanzId();

    server.laufendeProzesse(1);
    server.erwarteBenutzerFormular(prozessInstanzId, BESTELL_FORMULAR);

    Formular formular = new Formular();
    formular.setBestellDatum("01.01.2015");
    formular.setStandardVersand(false);
    String taskId = server.taskIdZu(prozessInstanzId);
    controller.submitEingabe(formular,
         Mockito.mock(BindingResult.class), null,
         taskId, null, ABBRUCH);

    server.laufendeProzesse(0);
}
```

— "Testskript"

— "Incidental Details"

„Im Bestellformular soll der Benutzer mit dem Abbrechen-Button den Bestellprozess beenden können"

```
@Test public void prozessAbbruch_inBestellFormular() {
    server.laufendeProzesse(0);
    ProzessStatus prozess = server.starteProzess(Prozess.BESTELLUNG);
    prozessInstanzId = prozess.getInstanzId();

    server.laufendeProzesse(1);
    server.erwarteBenutzerFormular(prozessInstanzId, BESTELL_FORMULAR);

    Formular formular = new Formular();
    formular.setBestellDatum("01.01.2015");
    formular.setStandardVersand(false);
    String taskId = server.taskIdZu(prozessInstanzId);
    controller.submitEingabe(formular,
         Mockito.mock(BindingResult.class), null,
         taskId, null, ABBRUCH);

    server.laufendeProzesse(0);
}
```

Signal-Noise-Ratio?

Single Level of Abstraction?

„Im Bestellformular soll der Benutzer mit dem Abbrechen-Button den Bestellprozess beenden können"

```
@Test public void prozessAbbruch_inBestellFormular() {
    imFormular(BESTELL_FORMULAR);

    sendeFormularMitButton(ABBRUCH);

    prozessGestoppt();
}
```

Namen

Name Testklasse

```
public class ArOrderProcess {

   OrderProcess process;

   @Before public void createOrderProcess() {
      process = new OrderProcess();
   }
```

object under test

Name Testmethode

```
public class AnOrderProcess {

  OrderProcess process;

  @Before public void createOrderProcess() {
     process = new OrderProcess();
  }

  @Test public void inOrderForm_cancel() {
      process.setState(ORDER_FORM);

      process.submit(CANCEL_BUTTON);

      assertThat("process canceled", process.isCanceled(), equalTo(true));
  }
```

object under test

Namen

```
public class AnOrderProcess {

    OrderProcess process;

    @Before public void createOrderProcess() {
        process = new OrderProcess();
    }

    @Test public void inOrderForm_cancel() {
        process.setState(ORDER_FORM);

        process.submit(CANCEL_BUTTON);

        assertThat("process canceled", process.isCanceled(), equalTo(true));
    }
```

object under test

setup

Namen

```
@Before public void createOrderProcess() {
    process = new OrderProcess();
}

@Test public void inOrderForm_cancel() {
    process.setState(ORDER_FORM);

    process.submit(CANCEL_BUTTON);

    assertThat("process canceled", process.isCanceled(), equalTo(true));
}
```

setup

execute

Namen

```
@Before public void createOrderProcess() {
    process = new OrderProcess();
}

@Test public void inOrderForm_cancel() {
    process.setState(ORDER_FORM);

    process.submit(CANCEL_BUTTON);

    assertThat("process canceled", process.isCanceled(), equalTo(true));
}
```

setup

execute

verify

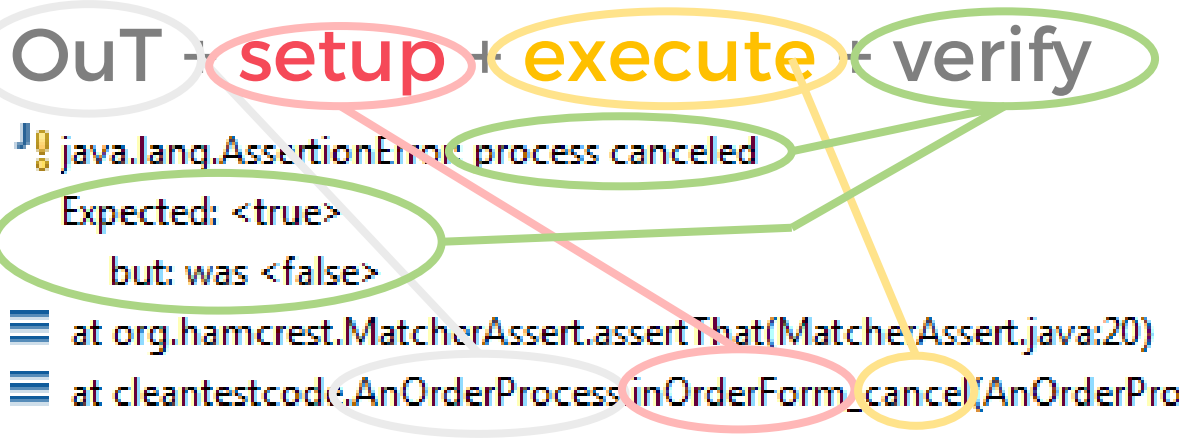
Publikum

- Test-Wartung
- Fehlersuche
- Spezifikation

Kommuniziere

vollständig

- OuT + setup + execute + verify



- Verteilung: Klasse, Methoden-Name, Assertion

Redundanzfrei

@Rule

```
public class BestellProzessTest {
    @Rule
    public ProzessTestServer server;
}

public class ProzessTestServer extends ExternalResource {
    @Override
    protected void before() throws Throwable {
         deployProcesses();
    }

    @Override
    protected void after() {
         deleteAllProcessInstances();
    }
```

Hierarchische Tests

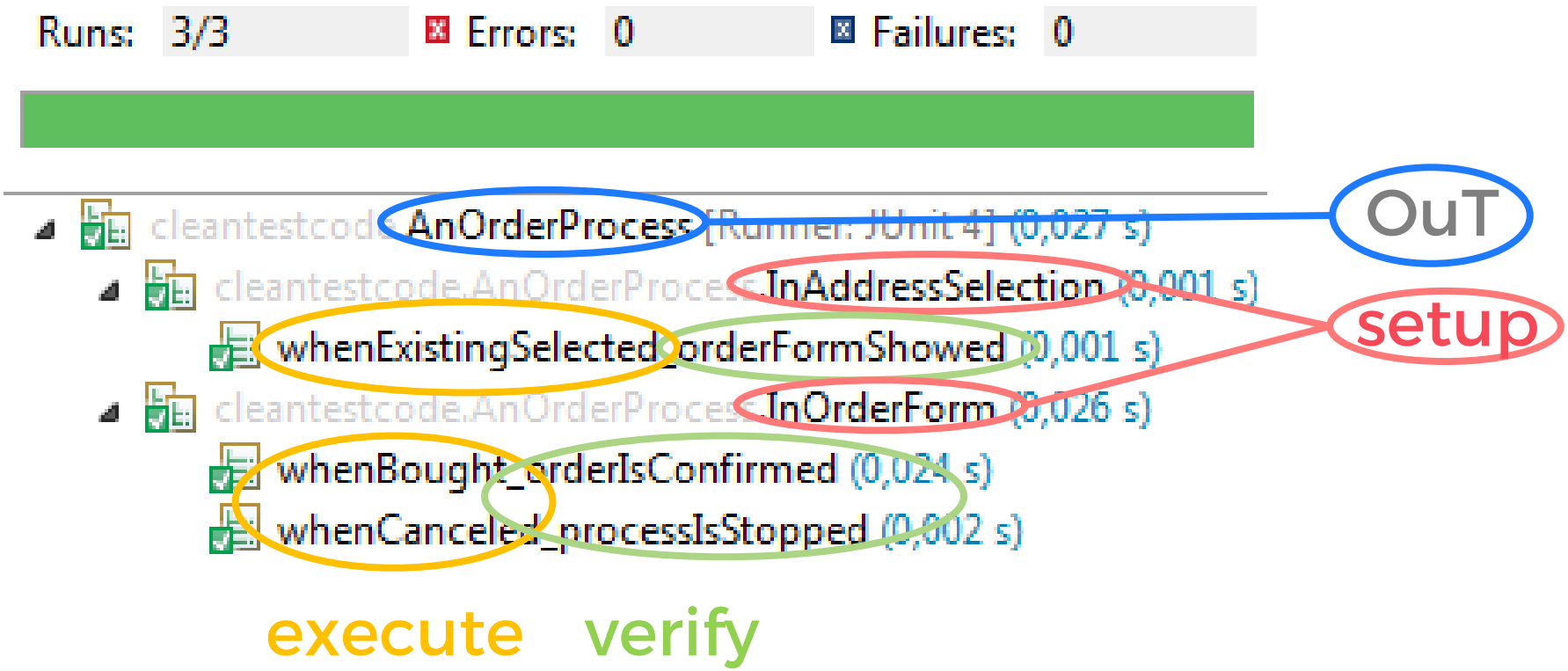
```
public class AnOrderProcess {

    @Before public void createOrderProcess() {
        process = new OrderProcess();
    }

    public class InOrderForm {
        @Before public void inOrderForm() {
            process.setState(ORDER_FORM);
        }
        @Test public void whenCanceled_processIsStopped() {
            process.submit(CANCEL_BUTTON);
            assertThat("process stopped", process.isStopped(), equalTo(true));
        }
        @Test public void whenBought_orderIsConfirmed() {
            process.submit(BUY_NOW_BUTTON);
            assertThat("state", process.getState(), equalTo(ORDER_CONFIRMED));
        }
    }
    …
```

gemeinsames
“setup”

Hierarchische Tests



Gemeinsame Testdaten

- Hilfsmethoden

```
aCustomer("David");
CustomerTest.aCustomer("David");
```

- Object Mother

```
CustomerMother.aCustomer("David");
CustomerMother.aCustomer("David", „Völkel");
CustomerMother.aCustomer(null, „Völkel", null,
        null, null, null, birthday);
```

- Test Data Builder

```
aCustomer().withLastName("Völkel")
           .withBirthDay(birthDay)
           .build();
```

Gemeinsames Verify

- Hilfsmethoden

```
void assertMoneyEquals(Money expected, Money actual) {
    assertEquals("currency", expected.getCurrency(),actual.getCurrency());
    assertEquals("amount", expected.getAmount(), actual.getAmount());
}
```

- Komposition via Hamcrest Matcher

```
Matcher<Money> equalTo(Money money) {
    return allOf(
        property(money, Money::getCurrency, "currency"),
        property(money, Money::getAmount, "amount"));
}
```

Fokussiert

"Single Concept per Test"

- "One Execute per Test"
- "One Assert per Test"

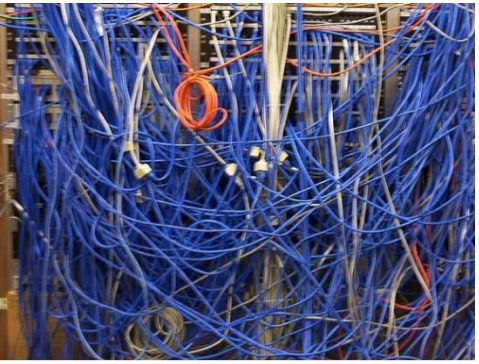
Fokussiert

Parameterized „State of Java“

```
@Test public void beitragsfreiheit() {
    freigabeZustandIst(2, BEITRAGS_FREI);
    freigabeZustandIst(3, BEITRAGS_FREI);
    freigabeZustandIst(99, !BEITRAGS_FREI);
    freigabeZustandIst(null, !BEITRAGS_FREI);
}
```

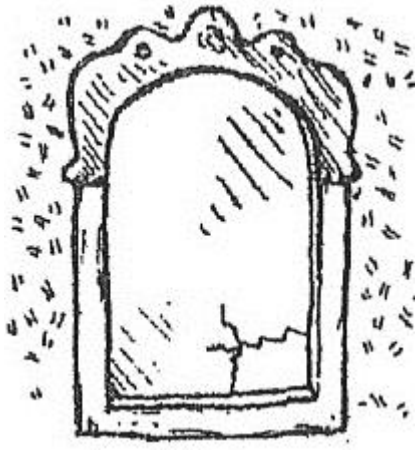
Fokussiert

Status vs. Testbarkeit



Hard to test code

„Listen to your tests!“



Clean Tests

lesbar

wartbar ← flexibel

verlässlich

Clean Tests 1.

lesbar

wartbar ← flexibel

1. verlässlich

Clean Tests

2. lesbar ← Namen & Abstraktion

wartbar ← flexibel

verlässlich

Clean Tests

lesbar

Namen &
Abstraktion

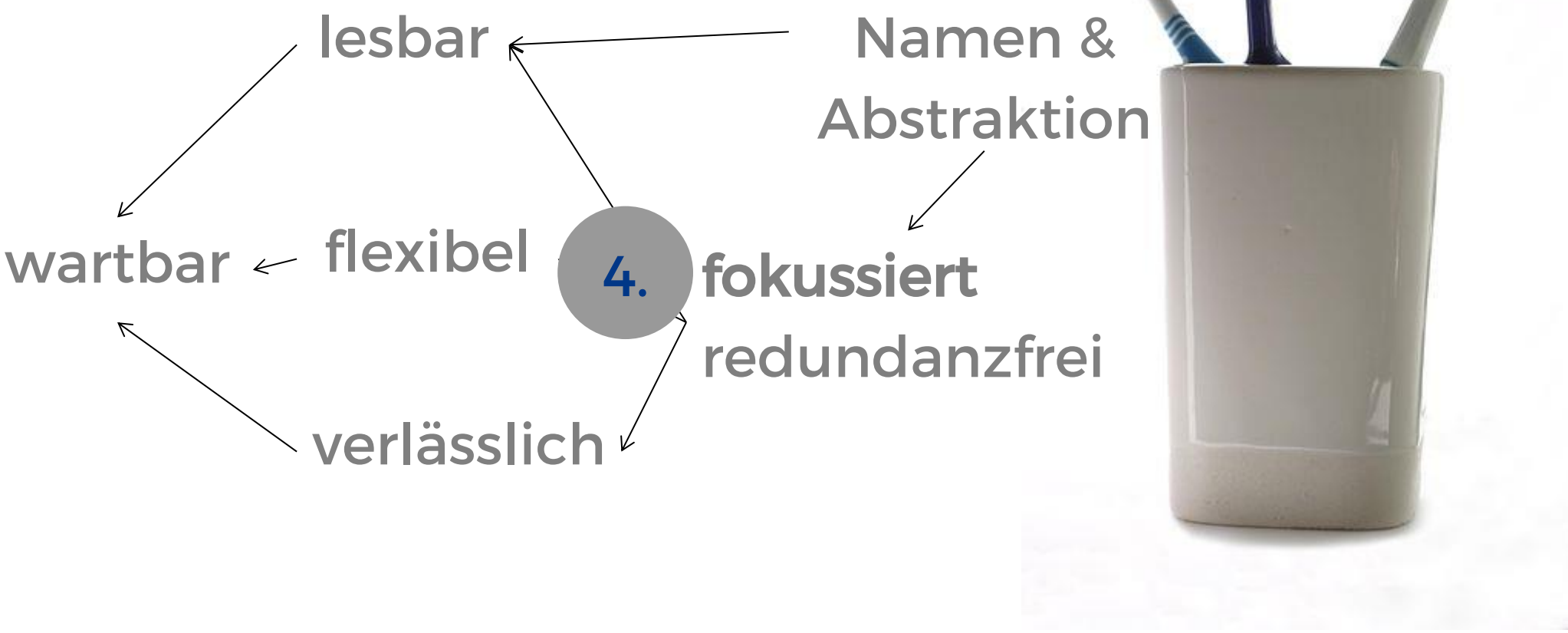
wartbar

flexibel

3. redundanzfrei

verlässlich

Clean Tests


lesbar
Namen &
Abstraktion
wartbar
flexibel
4.
fokussiert
redundanzfrei
verlässlich

Fazit

Agil bleiben mit
Clean Test Code

Quellen

Bücher

- „Clean Code“, Robert C. Martin
- „xUnit Test Patterns“, Gerard Meszaros
- „Effective Unittesting“, Lasse Koskela
- „Growing Object Oriented Software“, Steve Freeman, Nat Pryce
- „Specification by Example”, Gojko Adzic

Videos

- Episoden 20-22 http://cleancoders.com/, Robert C. Martin

Artikel

- „Design – jetzt auch für Tests“, David Völkel

codecentric
agile software factory

Fragen und Diskussion

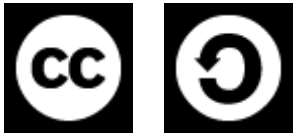
David Völkel
codecentric AG

Twitter: @davidvoelkel
david.voelkel@codecentric.de

www.codecentric.de
blog.codecentric.de

Lizenz

- Attribution-ShareAlike 3.0 Germany
- http://creativecommons.org/licenses/by-sa/3.0/de/



- Die verwendeten Bilder sind gemeinfrei und stammen aus der Wikipedia