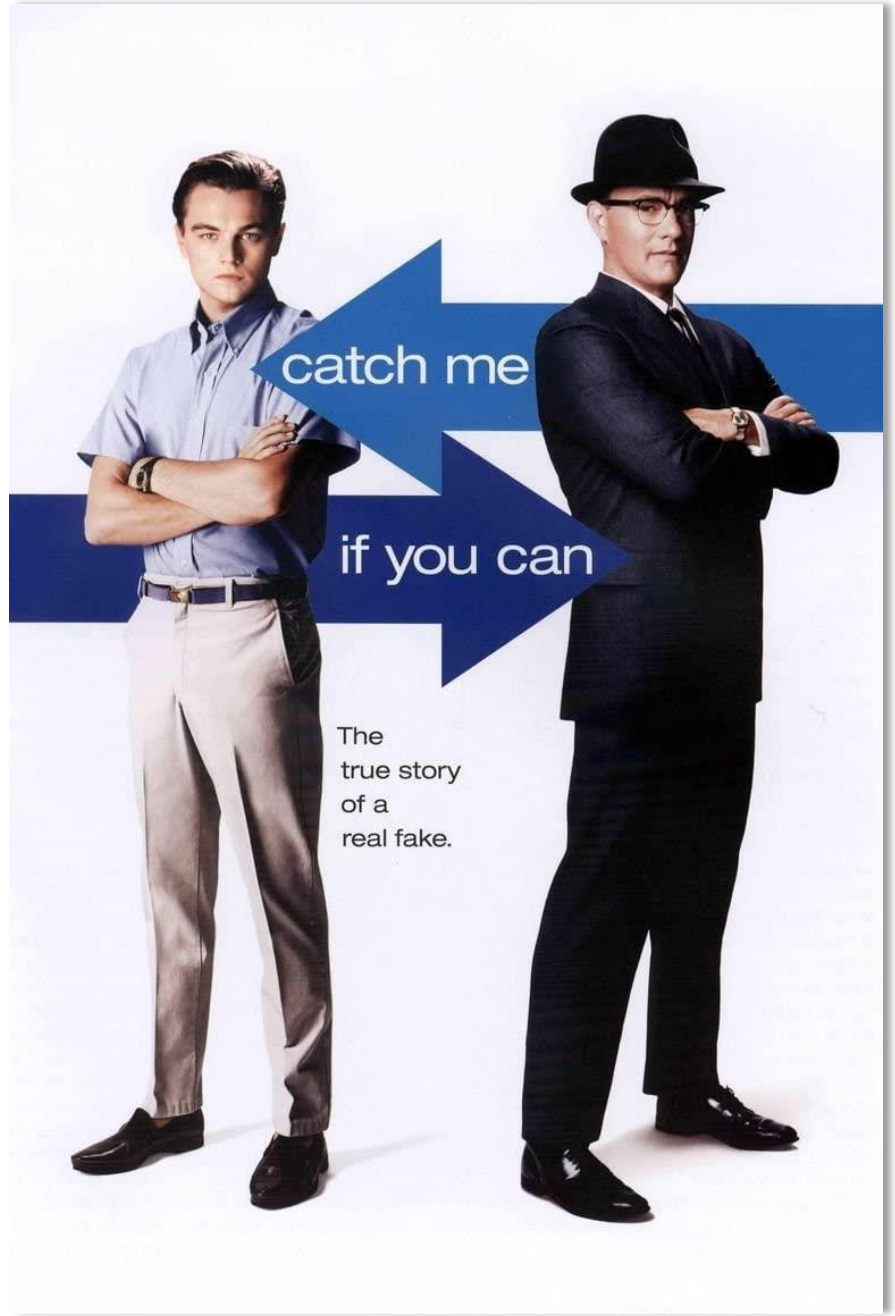
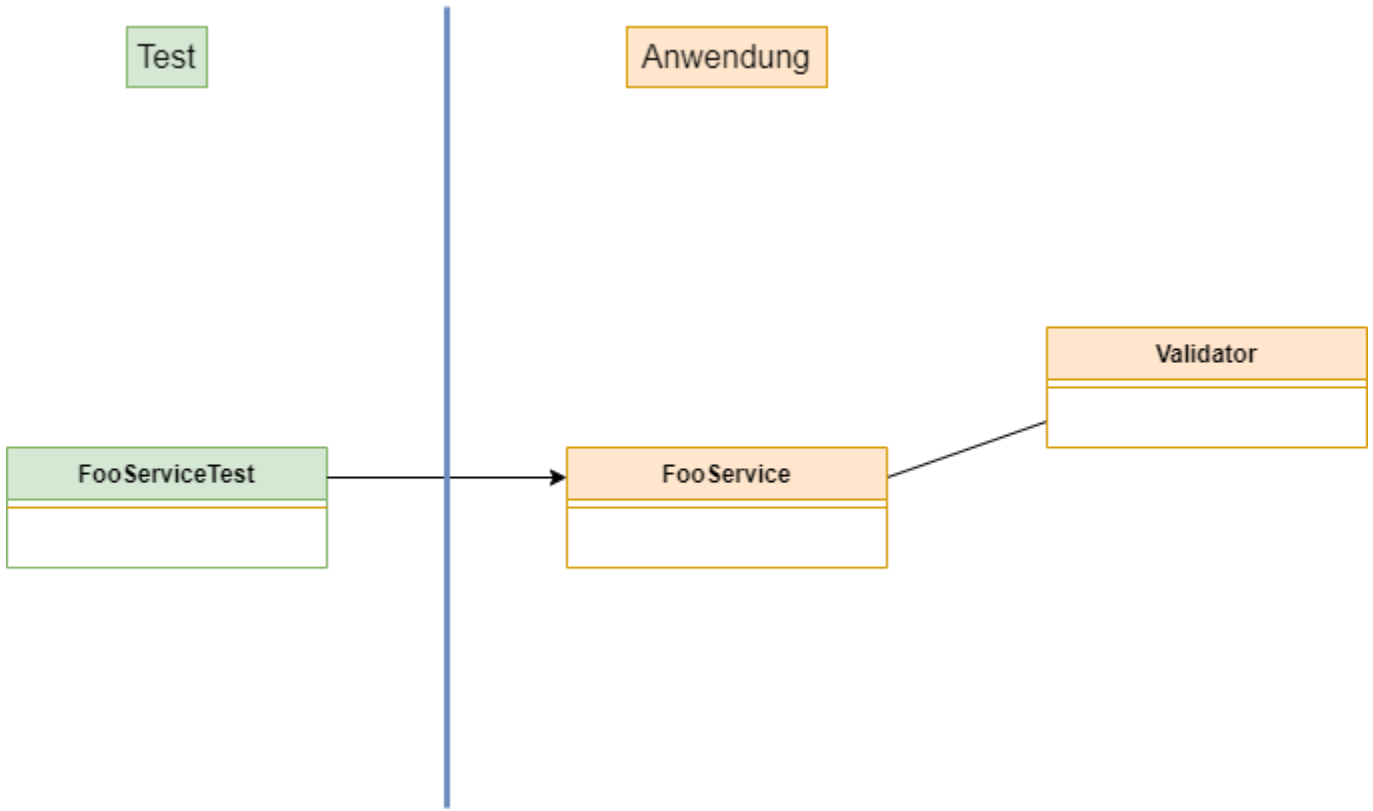
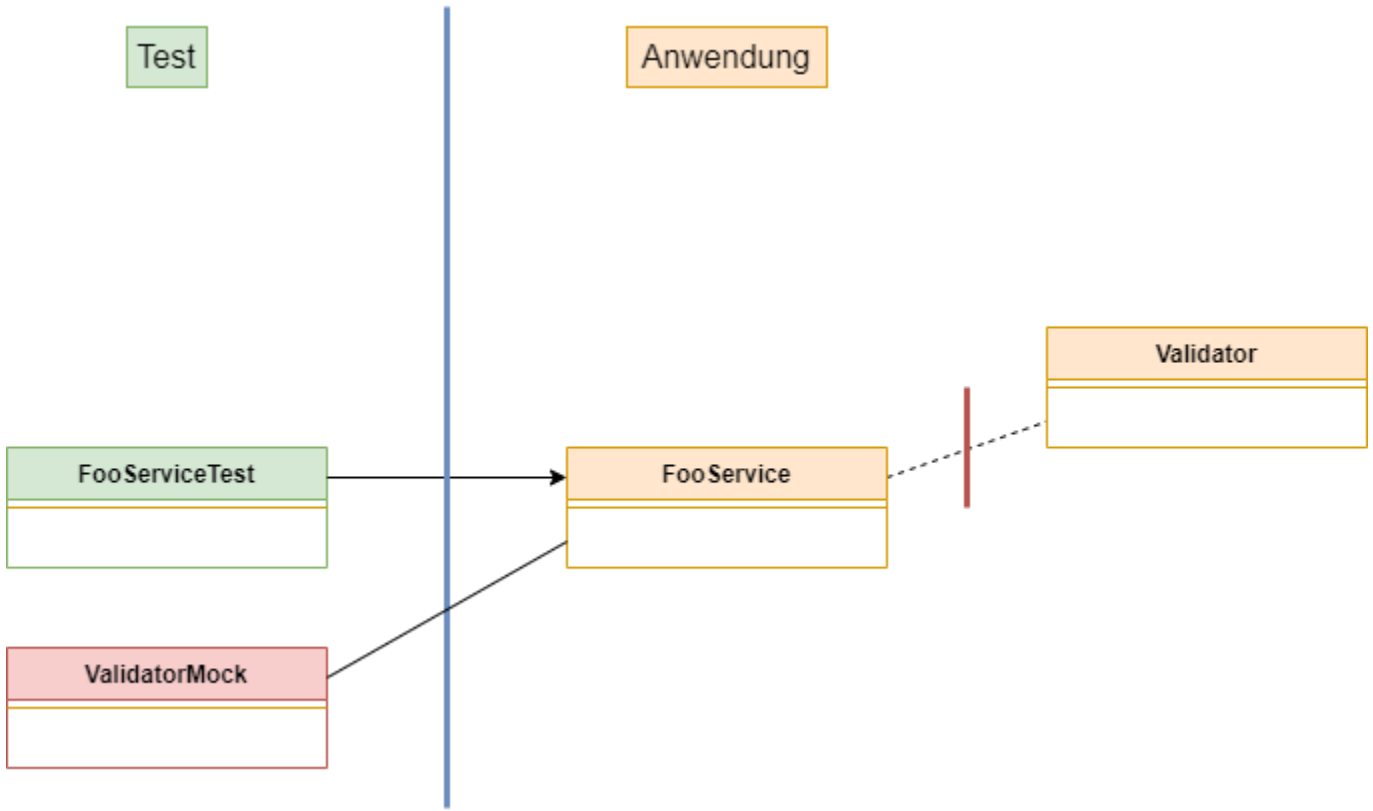
doubleSlash

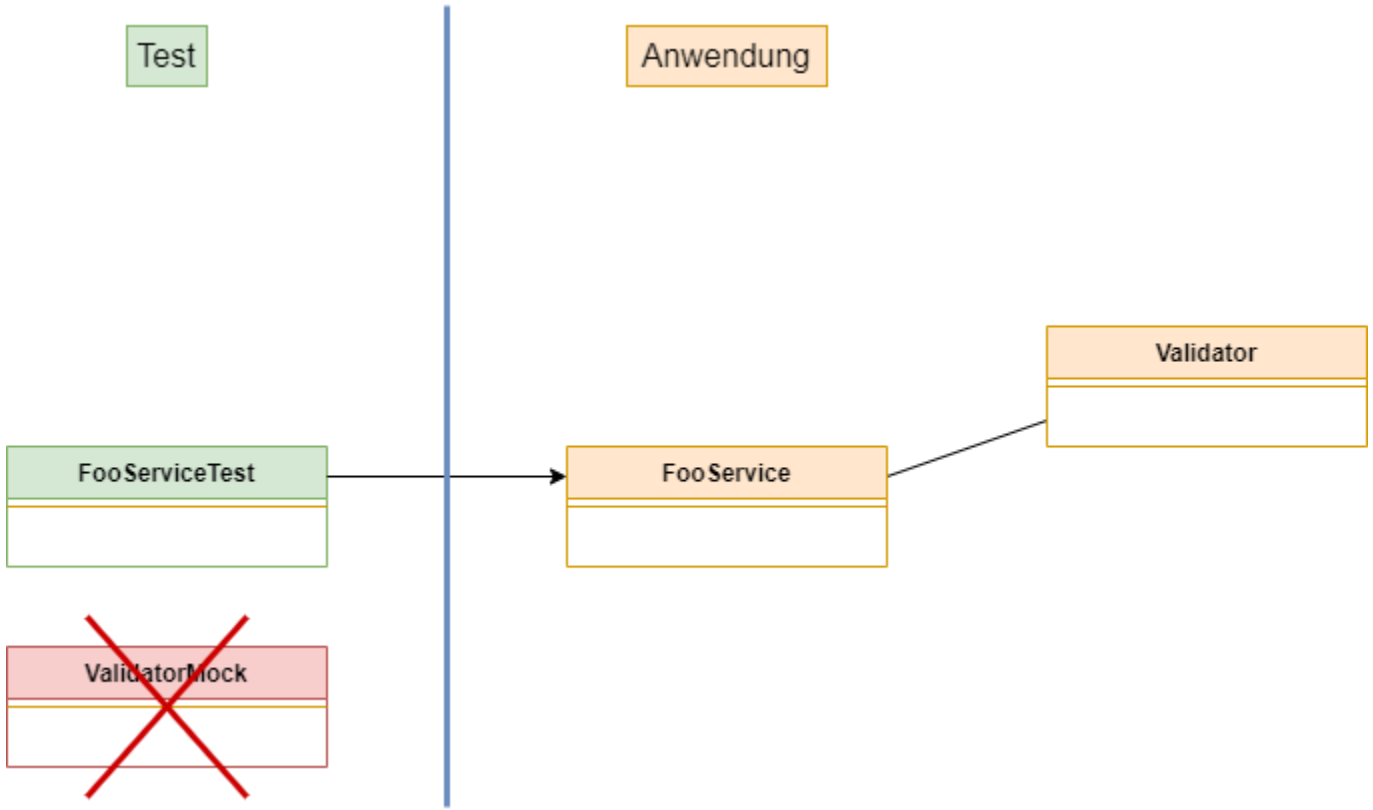
JUnit Tests? Exzellent!

Die Eigenschaften guter Tests


catch me
if you can
The
true story
of a
real fake.


Test
Anwendung
Validator
FooServiceTest
FooService


Test
Anwendung
Validator
FooServiceTest
FooService
ValidatorMock


Test
Anwendung
Validator
FooServiceTest
FooService
ValidatorMock

Gute Tests sind SINNVOLL

→ keine Anwendungslogik mocken

Gute Tests sind SINNVOLL

→ keine Anwendungslogik mocken
→ nur für Coverage?

Gute Tests sind SINNVOLL

→ keine Anwendungslogik mocken
→ „Risk Coverage“

Gute Tests sind SINNVOLL

→ keine Anwendungslogik mocken
→ „Risk Coverage“
→ getter u. setter?

Gute Tests sind SINNVOLL

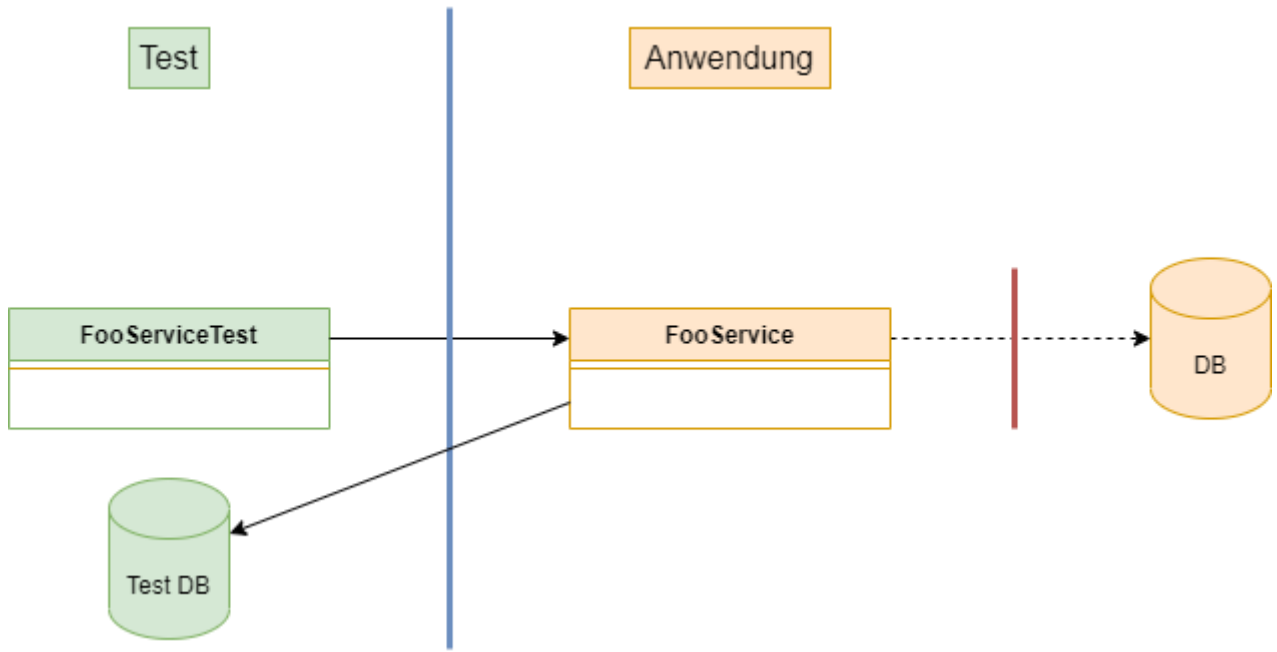
→ keine Anwendungslogik mocken
→ „Risk Coverage“
→ getter u. setter?
→ deaktivierte Tests?

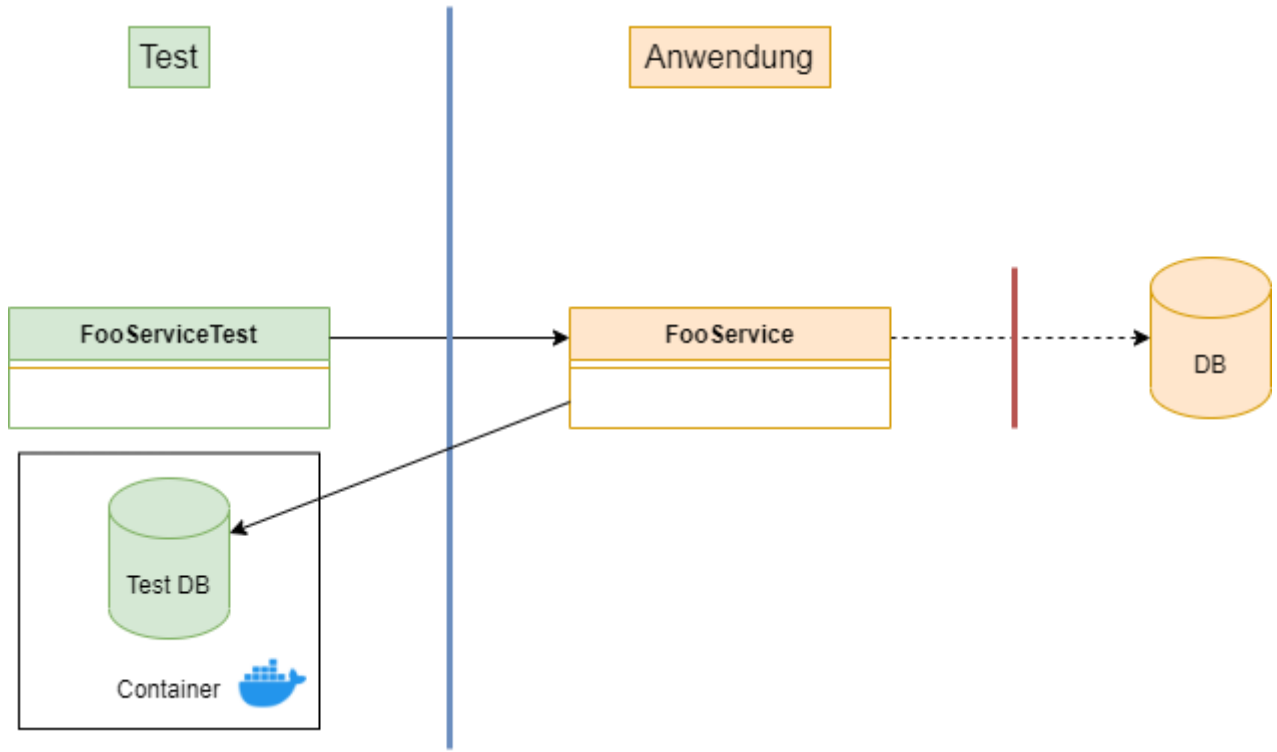

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

Gute Tests sind ZUVERLÄSSIG

= unabhängig von

→ Zeit

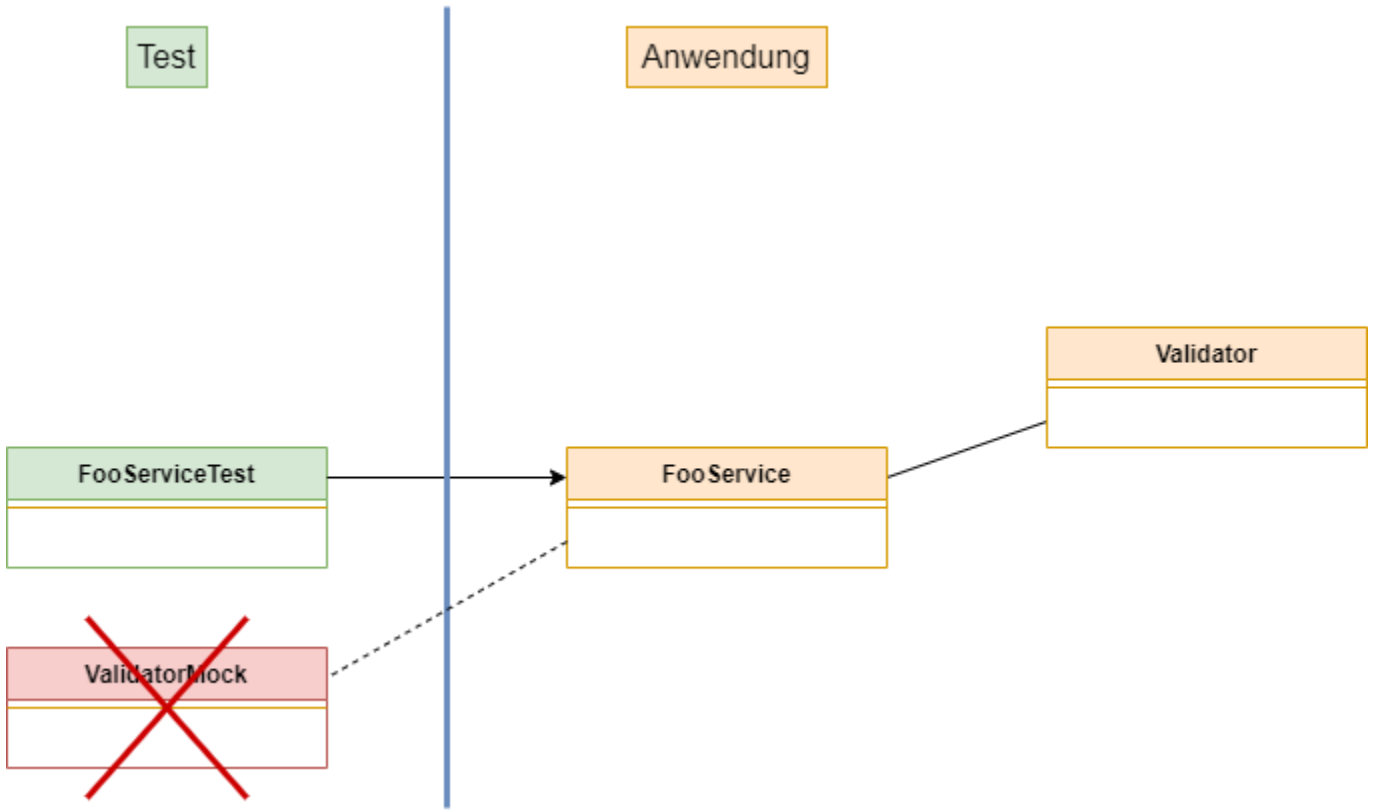

Test
Anwendung
FooServiceTest
FooService
DB
Test DB


Test
Anwendung
FooServiceTest
FooService
DB
Test DB
Container

Gute Tests sind ZUVERLÄSSIG

= unabhängig von

→ Zeit
→ anderen Tests (kein „shared state“)
→ Abhängigkeiten des SUT?


Test
Anwendung
Validator
FooServiceTest
FooService
ValidatorMock

„A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system. […]“

Quelle: What Is Unit Testing (smartbear.com)


VERGESSEN DU MUSST
WAS FRÜHER DU GELERNT
imgflip.com

„Für Kent Beck ist [ein Unit Test] ein Test der »isoliert von anderen Tests« läuft.

Die Einheit für die Isolation ist der Test, nicht die Klasse. […] Es ist die Isolation der Tests die uns antreibt, nicht die Isolation des Codes den wir testen.“

TDD, Where Did It All Go Wrong (Ian Cooper)

Quelle: TDD, Where Did It All Go Wrong

Gute Tests sind ZUVERLÄSSIG

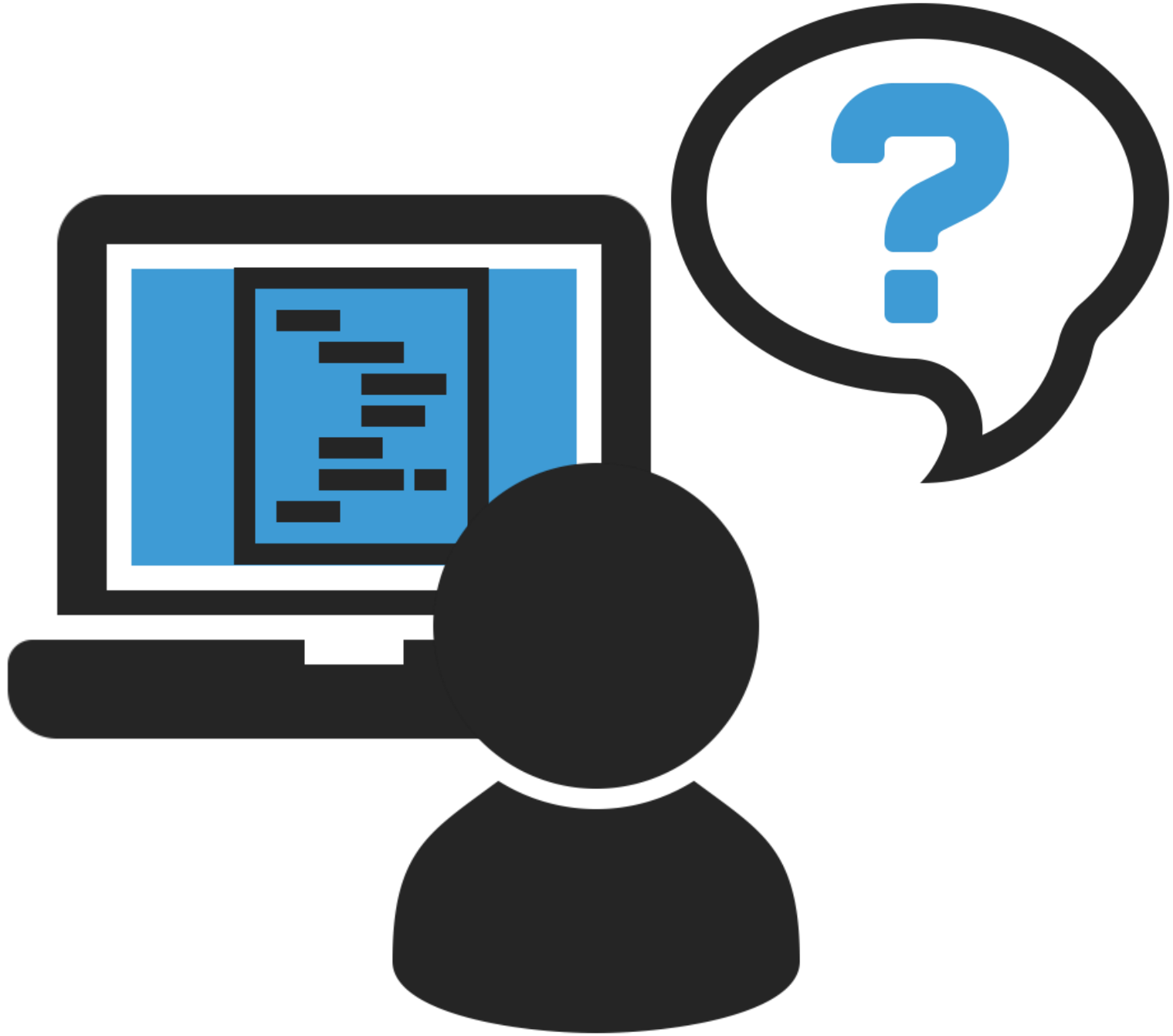
= isoliert von

→ Zeit
→ anderen Tests (kein „shared state“)
~~→ Abhängigkeiten des SUT?~~

Gute Tests sind ZUVERLÄSSIG

= isoliert von

→ Zeit
→ anderen Tests („kein shared state“)
→ Laufzeitumgebung
→ unzuverlässigen / langsamen Abhängigkeiten



```java
@Test
void test() {
    Account a = new Account();
    a.setAccountNumber(11111);
    a.setBalance(50.0);
    a.setLineOfCredit(1000.0);
    Account b = new Account();
    b.setAccountNumber(22222);
    b.setBalance(120.0);
    b.setLineOfCredit(500.0);
    BankingService bankingService = new BankingService();
    bankingService.transfer(a, b, 100);
    assertEquals(11111, a.getAccountNumber());
    assertEquals(1000.0, a.getLineOfCredit());
    assertEquals(a.getBalance(), -50.0);
    assertEquals(22222, b.getAccountNumber());
    assertEquals(500.0, b.getLineOfCredit());
    assertEquals(220.0, b.getBalance());
}
```

```java
@Test
void test() {
    // arrange
    Account a = new Account();
    a.setAccountNumber(11111);
    a.setBalance(50.0);
    a.setLineOfCredit(1000.0);
    Account b = new Account();
    b.setAccountNumber(22222);
    b.setBalance(120.0);
    b.setLineOfCredit(500.0);
    BankingService bankingService = new BankingService();

    // act
    bankingService.transfer(a, b, 100);

    // assert
    assertEquals(11111, a.getAccountNumber());
    assertEquals(1000.0, a.getLineOfCredit());
    assertEquals(a.getBalance(), -50.0);
    assertEquals(22222, b.getAccountNumber());
    assertEquals(500.0, b.getLineOfCredit());
    assertEquals(220.0, b.getBalance());
}
```

```java
@Test
void test() {
    // arrange
    Account a = new Account();
    a.setAccountNumber(11111);
    a.setBalance(50.0);
    a.setLineOfCredit(1000.0);
    Account b = new Account();
    b.setAccountNumber(22222);
    b.setBalance(120.0);
    b.setLineOfCredit(500.0);
    BankingService bankingService = new BankingService();

    // act
    bankingService.transfer(a, b, 100);

    // assert
    assertEquals(a.getBalance(), -50.0);
    assertEquals(220.0, b.getBalance());
}
```

```java
@Test
void test() {
    // arrange
    Account a = new Account();
    a.setAccountNumber(11111);
    a.setBalance(50.0);
    a.setLineOfCredit(1000.0);
    Account b = new Account();
    b.setAccountNumber(22222);
    b.setBalance(120.0);
    b.setLineOfCredit(500.0);
    BankingService bankingService = new BankingService();

    // act
    bankingService.transfer(a, b, 100);

    // assert
    assertThat(a.getBalance()).isEqualTo(-50.0);
    assertThat(b.getBalance()).isEqualTo(220.0);
}
```

```java
@Test
void test() {
    // arrange
    Account a = AccountBuilder.testAccount()
            .withBalance(50.0).build();
    Account b = AccountBuilder.testAccount()
            .withBalance(120.0).build();
    BankingService bankingService = new BankingService();

    // act
    bankingService.transfer(a, b, 100);

    // assert
    assertThat(a.getBalance()).isEqualTo(-50.0);
    assertThat(b.getBalance()).isEqualTo(220.0);
}
```

```java
@Test
void shouldTransferMoneyFromOneAccountToAnother() {
    // arrange
    Account fromAccount = AccountBuilder.testAccount()
            .withBalance(50.0).build();
    Account toAccount = AccountBuilder.testAccount()
            .withBalance(120.0).build();
    BankingService bankingService = new BankingService();

    // act
    bankingService.transfer(fromAccount, toAccount, 100);

    // assert
    assertThat(fromAccount.getBalance()).isEqualTo(-50.0);
    assertThat(toAccount.getBalance()).isEqualTo(220.0);
}
```

```java
public class BankingServiceTest {

    private final BankingService bankingService = new BankingService();

    @Test
    void shouldTransferMoneyFromOneAccountToAnother() {
        // arrange
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build();
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build();

        // act
        bankingService.transfer(fromAccount, toAccount, 100);

        // assert
        assertThat(fromAccount.getBalance()).isEqualTo(-50.0);
        assertThat(toAccount.getBalance()).isEqualTo(220.0);
    }
}
```

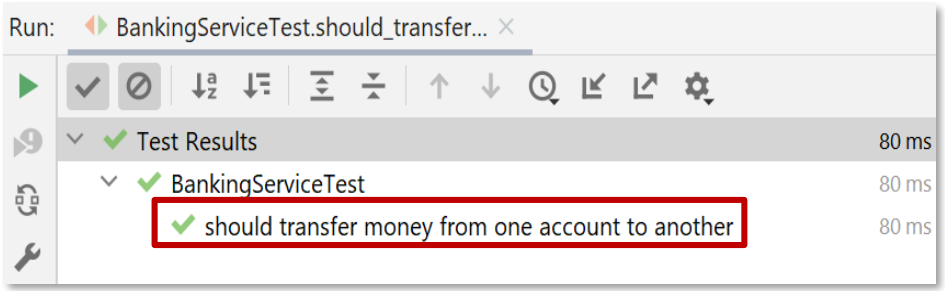
```java
@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)
public class BankingServiceTest {

    private final BankingService bankingService = new BankingService();

    @Test
    void should_transfer_money_from_one_account_to_another() {
        // arrange
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build();
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build();

        // act
        bankingService.transfer(fromAccount, toAccount, 100);

        // assert
        assertThat(fromAccount.getBalance()).isEqualTo(-50.0);
        assertThat(toAccount.getBalance()).isEqualTo(220.0);
    }
}
```



Let‘s talk about…

DSL‘s

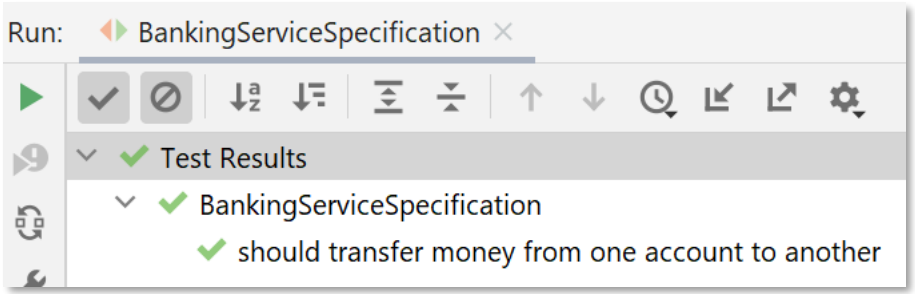
```groovy
class BankingServiceSpecification extends Specification {

    def BankingService bankingService = new BankingService()

    def "should transfer money from one account to another"() {
        given:
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build()
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build()

        when:
        bankingService.transfer(fromAccount, toAccount, 100)

        then:
        fromAccount.getBalance() == -50.0d
        toAccount.getBalance()   == 220.0d
    }
}
```



```groovy
class BankingServiceSpecification extends Specification {

    def BankingService bankingService = new BankingService()

    def "should transfer money from one account to another"() {
        given:
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build()
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build()

        when:
        bankingService.transfer(fromAccount, toAccount, 100)

        then:
        fromAccount.getBalance() == -50.0d
        toAccount.getBalance()   == 220.0d
    }
}
```

```groovy
class BankingServiceSpecification extends Specification {

    def BankingService bankingService = new BankingService()

    def "should transfer money from one account to another"() {
        given:
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build()
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build()

        when:
        bankingService.transfer(fromAccount, toAccount, 100)

        then:
        fromAccount.getBalance() == -50.0d
        toAccount.getBalance()   == 220.0d
    }
}
```

```groovy
class BankingServiceSpecification extends Specification {

    def BankingService bankingService = new BankingService()

    def "should transfer money from one account to another"() {
        given:
        Account fromAccount = AccountBuilder.testAccount()
                .withBalance(50.0).build()
        Account toAccount = AccountBuilder.testAccount()
                .withBalance(120.0).build()

        when:
        bankingService.transfer(fromAccount, toAccount, 100)

        then:
        fromAccount.getBalance() == -50.0d
        toAccount.getBalance()   == 220.0d
    }
}
```

```
Condition not satisfied:

toAccount.getBalance()   == 220.0d
|                        |
|                        false
|                        200.0
Account
```

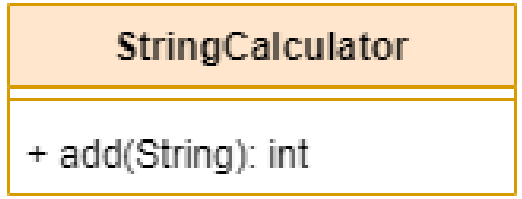
Zusammengefasst:

- Struktur (AAA / given, when, then)
- wesentliche Parameter im Test
- nur relevante Asserts
- sprechende Namen für Testmethoden
- Code durch DSLs sprechen lassen (AssertJ/Spock, Builder Pattern, Hilfsmethoden)
- aussagekräftige Fehlermeldungen

➔ verständliche Tests

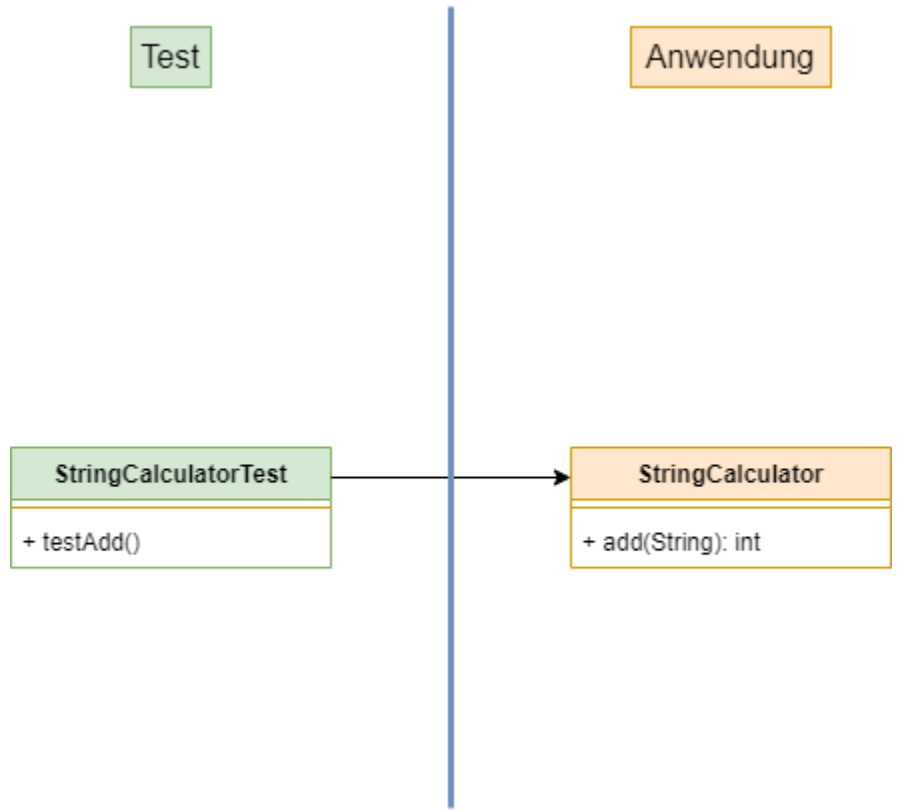
Gute Tests sind WARTBAR

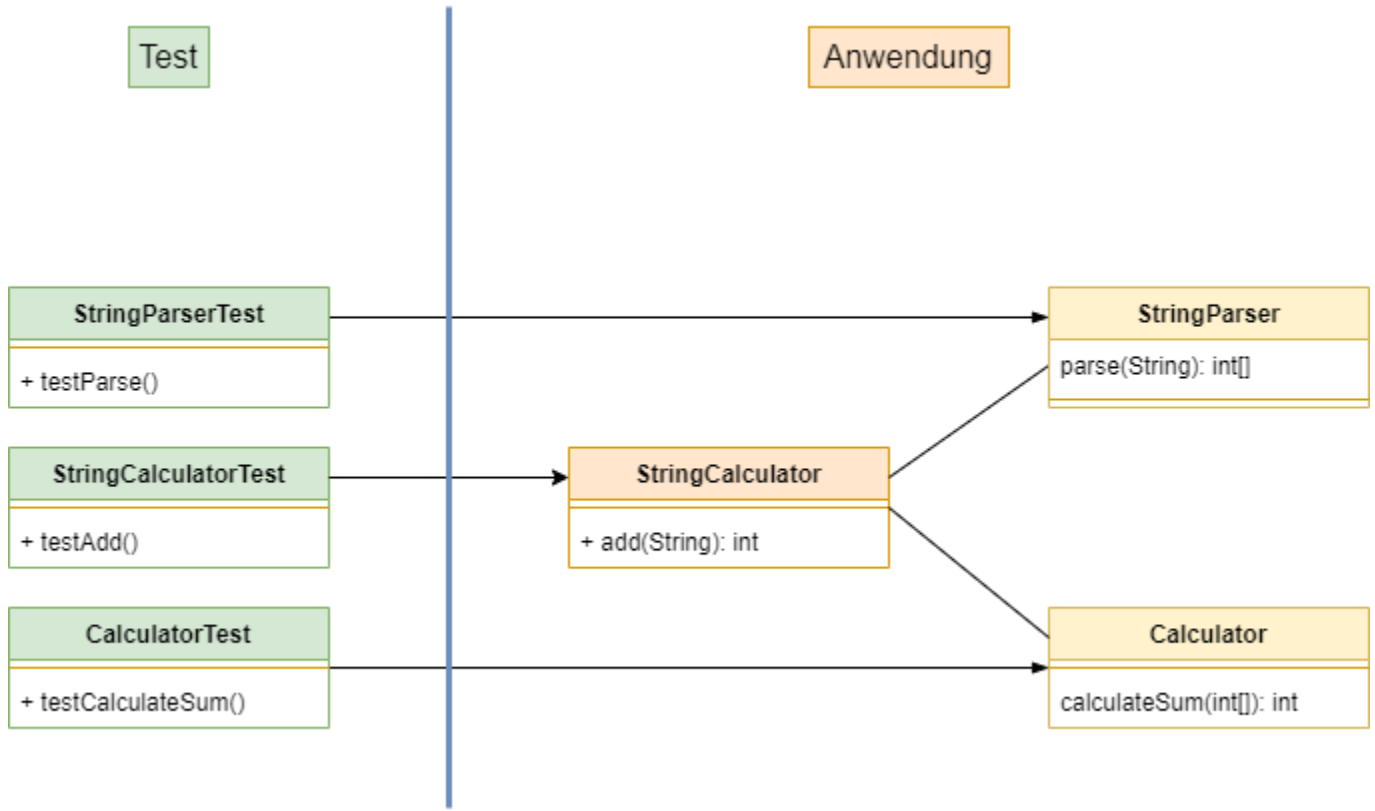
➔ verständlich



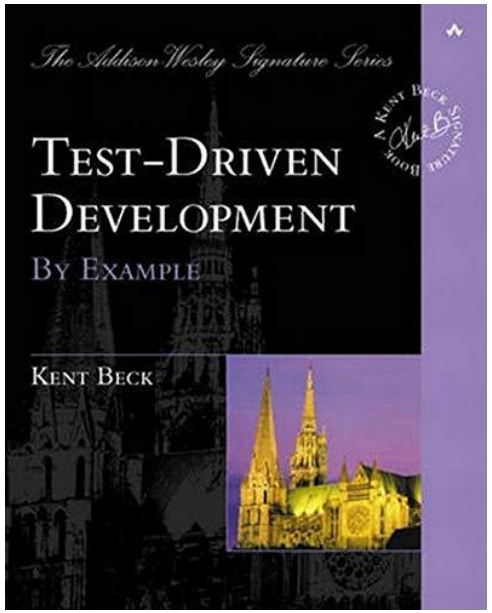

```
stringCalculator.add("");                  // -> 0
stringCalculator.add("1,2");               // -> 3
stringCalculator.add("1,10,100");          // -> 111
stringCalculator.add("//|\n1|2");          // -> 3
```

Quelle: [String Calculator TDD Kata]


Test
Anwendung
StringCalculatorTest
+ testAdd()
StringCalculator
+ add(String): int


Test
Anwendung
StringParserTest
+ testParse()
StringCalculatorTest
+ testAdd()
CalculatorTest
+ testCalculateSum()
StringCalculator
+ add(String): int
StringParser
parse(String): int[]
Calculator
calculateSum(int[]): int


VERGESSEN DU MUSST
WAS FRÜHER DU GELERNT
imgflip.com



„Welches Verhalten benötigen wir, um den überarbeiteten Bericht zu erstellen? Anders ausgedrückt: Welche Tests zeigen, wenn sie bestanden werden, das Vorhandensein von Code, bei dem wir sicher sind, dass er den Bericht korrekt berechnet?“

Test-Driven Development by Example (Kent Beck)

„Die Motivation für einen neuen Test ist eine neue Anforderung – nicht das Schreiben einer neuen Klasse oder Methode.“

TDD, Where Did It All Go Wrong (Ian Cooper)

Quelle: TDD, Where Did It All Go Wrong

„Die Struktur der Tests sollte kein Spiegelbild der Struktur des Anwendungscodes sein. Die Tatsache, dass es eine Klasse namens X gibt, sollte nicht automatisch bedeuten, dass es auch eine Testklasse namens XTest gibt. “

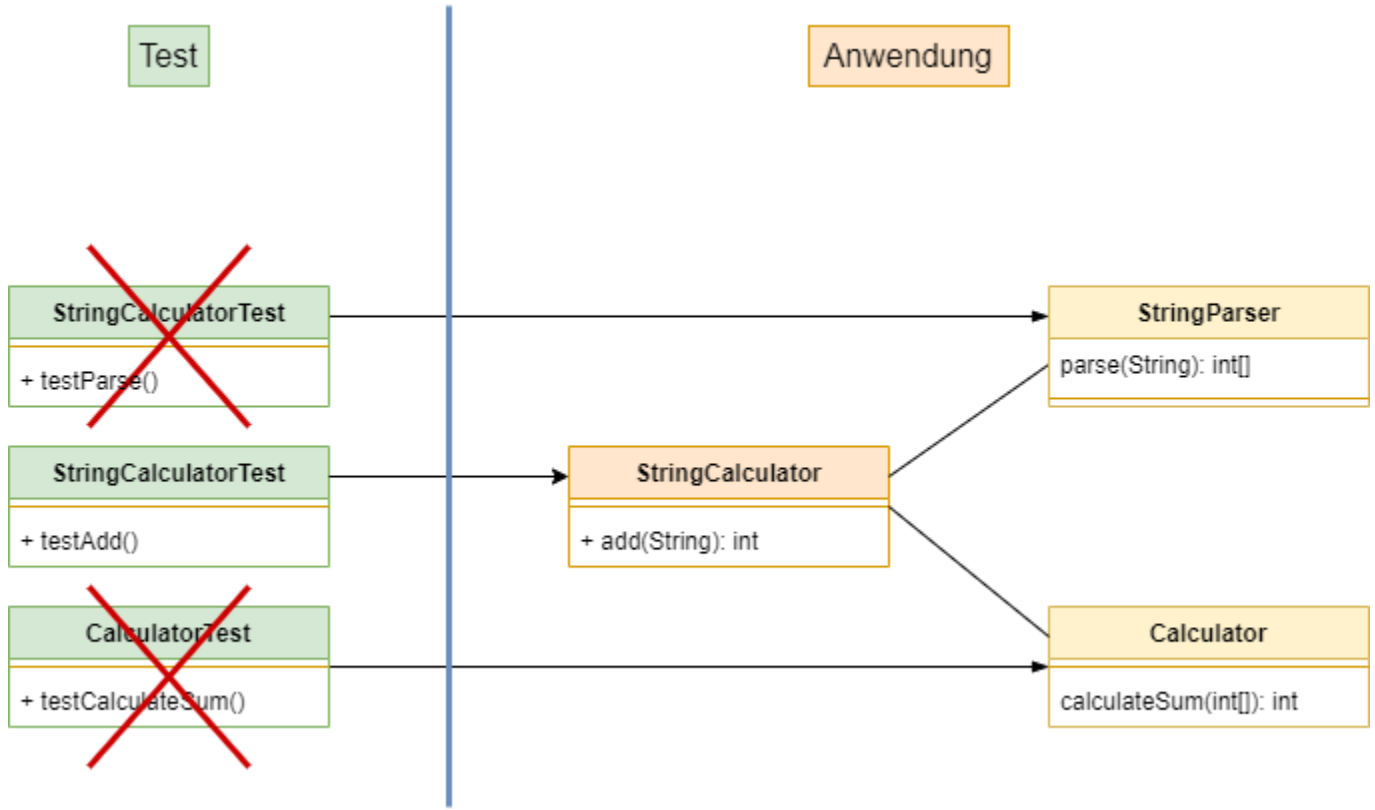
Robert C. Martin

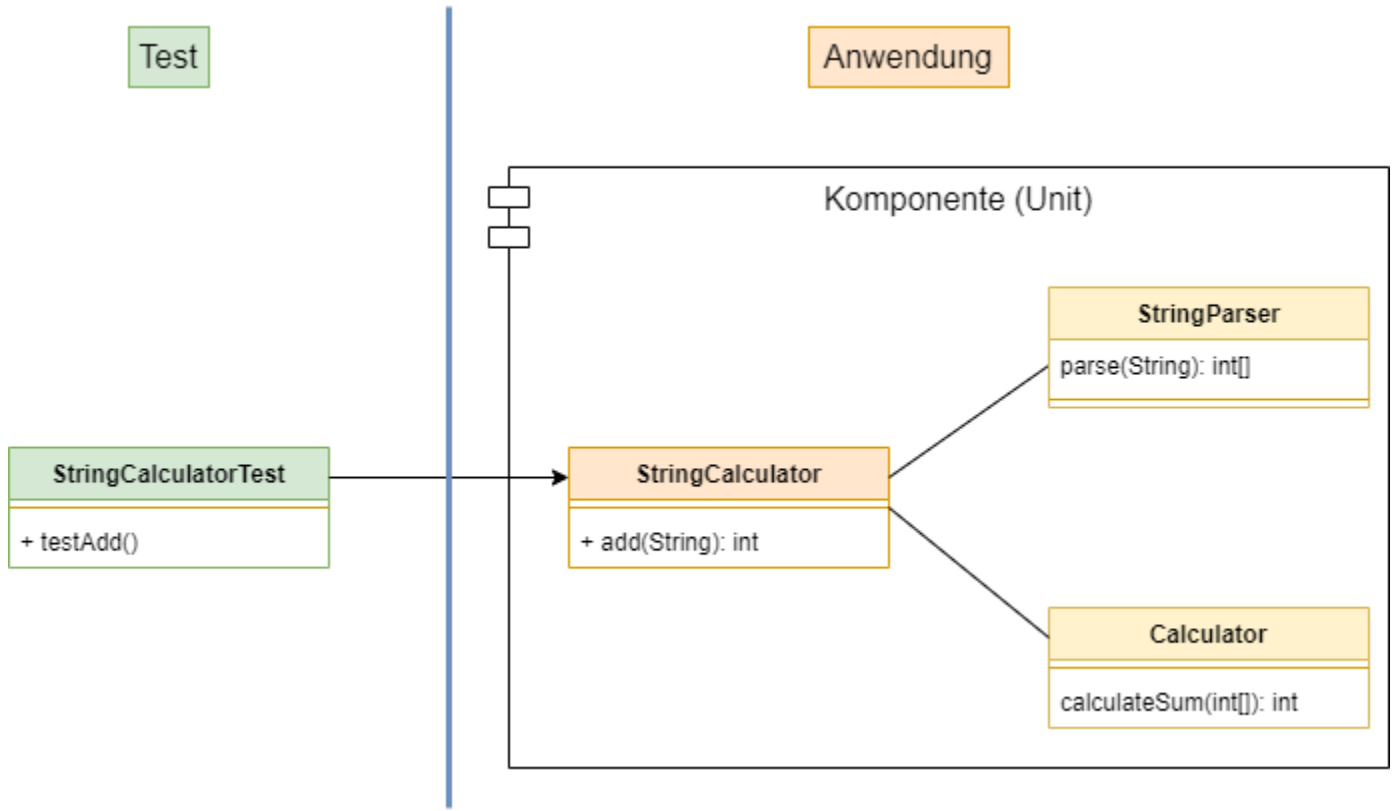
Quelle: Test Contra-variance, The Clean Code Blog

„Das Problem ist – und ich möchte dass Sie sorgfältig über die nächste Aussage nachdenken – ein 1:1-Verhältnis impliziert eine extrem enge Kopplung“.

Robert C. Martin

Quelle: TDD Harms Architecture, The Clean Code Blog


Test
Anwendung
StringCalculatorTest
+ testParse()
StringCalculatorTest
+ testAdd()
CalculatorTest
+ testCalculateSum()
StringCalculator
+ add(String): int
StringParser
parse(String): int[]
Calculator
calculateSum(int[]): int


Test
Anwendung
Komponente (Unit)
StringCalculatorTest
+ testAdd()
StringCalculator
+ add(String): int
StringParser
parse(String): int[]
Calculator
calculateSum(int[]): int

Gute Tests sind WARTBAR

→ verständlich
→ testen Verhalten,
keine Implementierungsdetails

Fazit

Gute Tests sind…

→ SINNVOLL
→ ZUVERLÄSSIG
→ WARTBAR

Vielen Dank!

// ✉ Stefan.Waldmann@doubleSlash.de

// https://blog.doubleslash.de/author/swaldmann/

// @waldmann42

Weiterführende Links

AssertJ – fluent assertions java library
https://assertj.github.io/doc/

Spock testing and specification framework
https://spockframework.org

Testcontainers
https://www.testcontainers.org/

Erzeugung von Testobjekten mit dem Builder Pattern
https://blog.doubleslash.de/lesbare-erzeugung-von-testobjekten-mit-dem-builder-pattern/

Modern Best Practices for Testing in Java
https://phauer.com/2019/modern-best-practices-testing-java/

Focus on Integration Tests Instead of Mock-based Tests
https://phauer.com/2019/focus-integration-tests-mock-based-tests/

Quellen

What is Unit Testing (smartbear.com)
https://smartbear.com/learn/automated-testing/what-is-unit-testing/

TDD, Where Did It All Go Wrong?
https://www.youtube.com/watch?v=EZ05e7EMOLM

Test Contra-variance (The Clean Code Blog)
https://blog.cleancoder.com/uncle-bob/2017/10/03/TestContravariance.html

TDD Harms Architecture (The Clean Code Blog)
https://blog.cleancoder.com/uncle-bob/2017/03/03/TDD-Harms-Architecture.html

Test-Driven Development by Example (Kent Beck)
https://www.amazon.de/Test-Driven-Development-Example-Signature/dp/0321146530